VISIT…

HTML Dog

The Best-Practice Guide to XHTML & CSS

Patrick Griffiths

New Riders

HTML Dog
Patrick Griffiths

New Riders
1249 Eighth Street
Berkeley, CA 94710
510/524-2178
800/283-9444
510/524-2221 (fax)

Find us on the Web at: www.newriders.com
To report errors, please send a note to errata@peachpit.com

New Riders is an imprint of Peachpit, a division of Pearson Education

Copyright © 2007 by Patrick Griffiths

Editor: **Doug Adrianson**

Production Coordinator: **Andrei Pasternak**

Tech Editor: **Joe Marini**

Copyeditor: **Hope Frazier**

Compositor: **Maureen Forys, Happenstance Type-O-Rama**

Indexer: **Julie Bess**

Cover Design: **Aren Howell**

Cover Photo: **Veer/Brian Summers**

Interior Design: **Maureen Forys Happenstance Type-O-Rama**

Notice of Rights

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. For information on getting permission for reprints and excerpts, contact permissions@peachpit.com.

Notice of Liability

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of the book, neither the author nor Peachpit Press shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it.

Trademarks

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Peachpit was aware of a trademark claim, the designations appear as requested by the owner of the trademark. All other product names and services identified throughout this book are used in editorial fashion only and for the benefit of such companies with no intention of infringement of the trademark. No such use, or the use of any trade name, is intended to convey endorsement or other affiliation with this book.

ISBN 0-321-31139-6

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

Acknowledgements

A good website follows conventions to keep users happy and responsive. I can only assume that a good web design book should do the same. So here are some people "without whom this would not have been possible." Or something like that...

To my mother, for her share of my genetic material and all of the environmental stuff, for buying me my first computer, for putting up with my Kevin & Perry teenage crap, and, most of all, for forbidding me to get a Michael Jackson perm at the age of 10, ta, Ma.

Even though her grasp of language is somewhat limited, for frequently walking across my keyboard Nutmeg, the feline member of the family, should probably have a co-author credit. At least blame any typos on her.

I am proud to be a member of such an open, intelligent, friendly professional community. Andy Budd, Andy Clarke, Jon Hicks, Jeremy Keith, Drew McLellan, Rich Rutter, Mike Stenhouse, and the rest of the Britpack (and the mighty Pub Standards, for that matter) have been an invaluable source of discussion, ideas, and constructive criticism, and have become good friends to boot. And there's a plethora of luminaries further from home who have influenced me, and this book, in one way or another: Doug Bowman, Dan Cederholm, Joe Clark, Charles Darwin, Molly Holzschlag, Steve Krug, Jakob Nielsen, Valentino Rossi, and Jeffrey Zeldman in particular. Through raising awareness, it's due to many of these people (and many more), and organizations like the Web Standards Project (*webstandards.org*) that the quality web design landscape is a much lusher one now than it was even a few years ago, so thanks are due not only for their influence, but for making books like this, and interest in them, possible.

Dan Webb (*danwebb.net*) has been the single most influential person when it comes to HTML Dog (site, book, and philosophy). From working together on numerous projects across the years to idle pub banter (across even more years), Dan is the first person I talked with about web standards, long before the emergence of that hat-wearing dude's little orange book, the person I have discussed around 43,082.6 aspects of web design with, from liquid layouts to accessibility to Microformats to the absurdity of the term *Web 2.0*, and the person who has proofread, edited, tested, and critiqued pretty much every single article and website that I have ever been involved in. Cheers, Dan.

I've had a little something to do with a bash called @media (*vivabit.com/atmedia*) for almost as long as the HTML Dog book project. Thanks to everyone who has made that possible, including all of those who have attended it. It has been a great example of a genuine appetite for pushing best-practice web design and development to their limits, and it has kept my enthusiasm and passion for the subject fresh. @media and HTML Dog are my babies, so they must be related.

I have always regarded New Riders as by far the best, most discerning, and most respectable publisher of Web-related books. It has been a roller-coaster ride, but I am very proud to finally be a published New Riders author alongside so many great Web heavyweights. So, to the publisher, and extended family and friends, thanks to David Fugate, Linda Bump Harrison, Darcy DiNucci, Marjorie Baer, Nancy Davis, Joe Marini, Doug Adrianson, and everyone else involved in building this quality culturally infused slab of ink-sprinkled reconstituted plant fibers.

—Patrick Griffiths
October 2006

Contents

Introduction xv

Chapter 1: Getting Started 1

HTML Syntax 1
- Elements, Tags, and Attributes 2
- Common Attributes 4
- The Basic Structure of an HTML Document 8
- The Generalist Tags—Div and Span 16

CSS Syntax 17
- Rules 17
- Selectors 18
- Properties 23
- Values 25
- Applying CSS to HTML 32

Chapter 2: Text 37

Structuring Text 37
- Basic Text Elements: Paragraphs, Line Breaks, and Emphasis 39
- Headings 40
- Quotations 42
- Abbreviations and Acronyms 43
- Preformatted Text and Computer Code 44
- Editorial Insertions and Deletions 46
- Multilanguage and Bidirectional Text 47
- Addresses 47

Styling Text 48
Fonts 48
Color 50
Size 50
Line Height 53
Bold and Italics 54
Upper and Lower Case 55
The Font Shorthand Property 55
Underline and Strikethrough 56
Letter and Word Spacing 57
Indenting 58
Horizontal Alignment 58
Vertical Alignment 59
More Text Styling Techniques 60

Chapter 3: Links 61

Anchor Elements and Hypertext References 62
Page Anchors 63
Link States: Link, Visited, Hover, Focus, and Active 65
Accessible Links 67
Tabbing 67
Access Keys 68
Link Titles 70
Pop-ups 71
Adjacent Links 71
Skipping Navigation 72

Chapter 4: Images 75

The `img` Element 77
Image Maps 81
Background Images 82
Image Replacement: Providing Graphical Alternatives for Text 88

Chapter 5: Layout 93

The Box Model 94
Width and Height 95
Padding 97
Borders 98
Margin 100
The `display` Property 104
Positioning 107
Static 107
Relative 108
Absolute 108
Fixed 110
Floating 110
Sample Page Layouts 119
Creating Columns 120
Adding a Page Header 126
Adding a Footer 127
Putting It All Together 130

Chapter 6: Lists 135

Structuring Lists 136
Unordered and Ordered Lists 136
Definition Lists 138
Lists as Navigation 140
Presenting Lists 142
List Markers—Bullets, Numbers, and Images 142
Horizontal Lists 146

Chapter 7: Scripts & Objects 147

JavaScript and the DOM 147
The `script` Element 147

Event Attributes 148
Manipulating the DOM 149
Objects 150

Chapter 8: Tables 155

Basic Tables 156
Merging Cells 158
Captions 160
Grouping Rows 161
Targeting Columns 162
Accessibility Considerations with Tables 164
Summaries 164
Associating Headers to Cells 165
Associating Cells to Headers 165
Presenting Tables 167
Border Collapsing 167
Speedier Tables: the Fixed Layout Algorithm 169
Empty Cells 170

Chapter 9: Forms 171

Form Elements 173
Form Fields and Buttons: `input`, `textarea`, and `select` 174
The `name` Attribute 174
Putting Controls in Blocks 175
`input` 175
`textarea` 182
`select` 183
Fieldsets 185
Accessible Forms 186
Labels 186

Styling Form Fields 187
Borders 188
Fonts 189
Backgrounds 189

Chapter 10: Multiple Media 191

Screen-Readers 192
Mobile Devices 192
Print 193
A Sample Print Stylesheet 195
Applying Media-Specific CSS 195
The `media` Attribute 196
Separate or Cascading 204
`@media` 204
In Conclusion 205

Appendix A: XHTML Reference 207

Tags 207
<a> 209
<abbr></abbr> 210
<acronym></acronym> 210
<address></address> 211
<area /> 212
<base /> 213
<bdo></bdo> 214
<blockquote></blockquote> 214
<body></body> 215

 216
<button></button> 216
<caption></caption> 217
<cite></cite> 218

<code></code> 218
<col /> 219
<colgroup></colgroup> 220
<dd></dd> 221
<del></del> 222
<dfn></dfn> 222
 223
<dl></dl> 224
<dt></dt> 224
 225
<fieldset></fieldset> 225
<form></form> 226
<h1></h1>, <h2></h2>, <h3></h3>, <h4></h4>, <h5></h5>, <h6></h6> 227
<head></head> 228
<html></html> 229
<img /> 230
<input /> 231
<ins></ins> 232
<kbd></kbd> 233
<label></label> 233
<legend></legend> 234
 234
<link /> 235
<map></map> 236
<meta /> 237
<noscript></noscript> 238
<object></object> 239
 240
<optgroup></optgroup> 241
<option></option> 241

 242
<param /> 243
<pre></pre> 244
<q></q> 244
<samp></samp> 245
<script></script> 246
<select></select> 247
 248
 248
<style></style> 249
<table></table> 250
<tbody></tbody> 251
<td></td> 252
<textarea></textarea> 254
<tfoot></tfoot> 255
<th></th> 256
<thead></thead> 258
<title></title> 259
<tr></tr> 260
 261
<var></var> 262
Bad Tags 262
Rotten Attributes 264

Appendix B: CSS Reference 265

Pseudo-classes 265
:active 265
:first 266
:first-child 266
:focus 266
:hover 267

:lang 267
:left 267
:link 268
:right 268
:visited 268
Pseudo-elements 269
:after 269
:before 269
:first-letter 269
:first-line 270
At-rules 270
@import 270
@media 270
@page 271
Properties 271
background 272
background-attachment 272
background-color 273
background-image 274
background-position 275
background-repeat 275
border, border-top, border-right, border-bottom, border-left 276
border-collapse 277
border-color, border-top-color, border-right-color, border-bottom-color, border-left-color 278
border-spacing 278
border-style, border-top-style, border-right-style, border-bottom-style, border-left-style 279
border-width, border-top-width, border-right-width, border-bottom-width, border-left-width 281
bottom 281

caption-side 282
clear 283
clip 283
color. 284
content 285
counter-increment 286
counter-reset 286
cursor 287
direction 289
display. 289
empty-cells 291
float. 291
font 292
font-family. 293
font-size 293
font-style 294
font-variant 295
font-weight 295
height 296
left. 297
letter-spacing 297
line-height. 298
list-style 299
list-style-image 299
list-style-position 300
list-style-type 300
margin, margin-top, margin-right, margin-bottom, margin-left. 301
max-height 302
max-width 303
min-height. 303
min-width 304

orphans . . . 304
outline . . . 305
outline-color . . . 305
outline-style . . . 306
outline-width . . . 307
overflow . . . 307
padding, padding-top, padding-right, padding-bottom, padding-left . . . 308
page-break-after . . . 309
page-break-before . . . 309
page-break-inside . . . 310
position . . . 310
quotes . . . 311
right . . . 312
table-layout . . . 313
text-align . . . 313
text-decoration . . . 314
text-indent . . . 315
text-transform . . . 315
top . . . 316
unicode-bidi . . . 316
vertical-align . . . 317
visibility . . . 318
white-space . . . 319
widows . . . 320
width . . . 320
word-spacing . . . 321
z-index . . . 322

Index . . . 323

Introduction

THE BEST WAY to build web pages is with web-standards-compliant HTML and CSS. HTML lays the foundation by structuring the content, and then CSS dolls it up and presents the page.

Using them in the right way—with web standards—leads to web pages that are faster, more manageable, more cross-compatible, and more accessible than web pages built any other "old-school" way.

This book is designed to take you through these symbiotic languages, explaining how to use them the web-standard way, comprehensively covering the components that make up a web page and the technical details involved in making those components.

HTML Dog?

The HTML Dog (*www.htmldog.com*) first popped into the world in 2003. Its mission was to provide short and easy-to-follow guides in (X)HTML and CSS, following best practices from the ground up (rather than teaching old-school methods first and then moving on to the right way of doing things), which no other resource did, and few do even now. Since then the website has grown both in size and popularity, and is now one of the web's most-used resources for web designers.

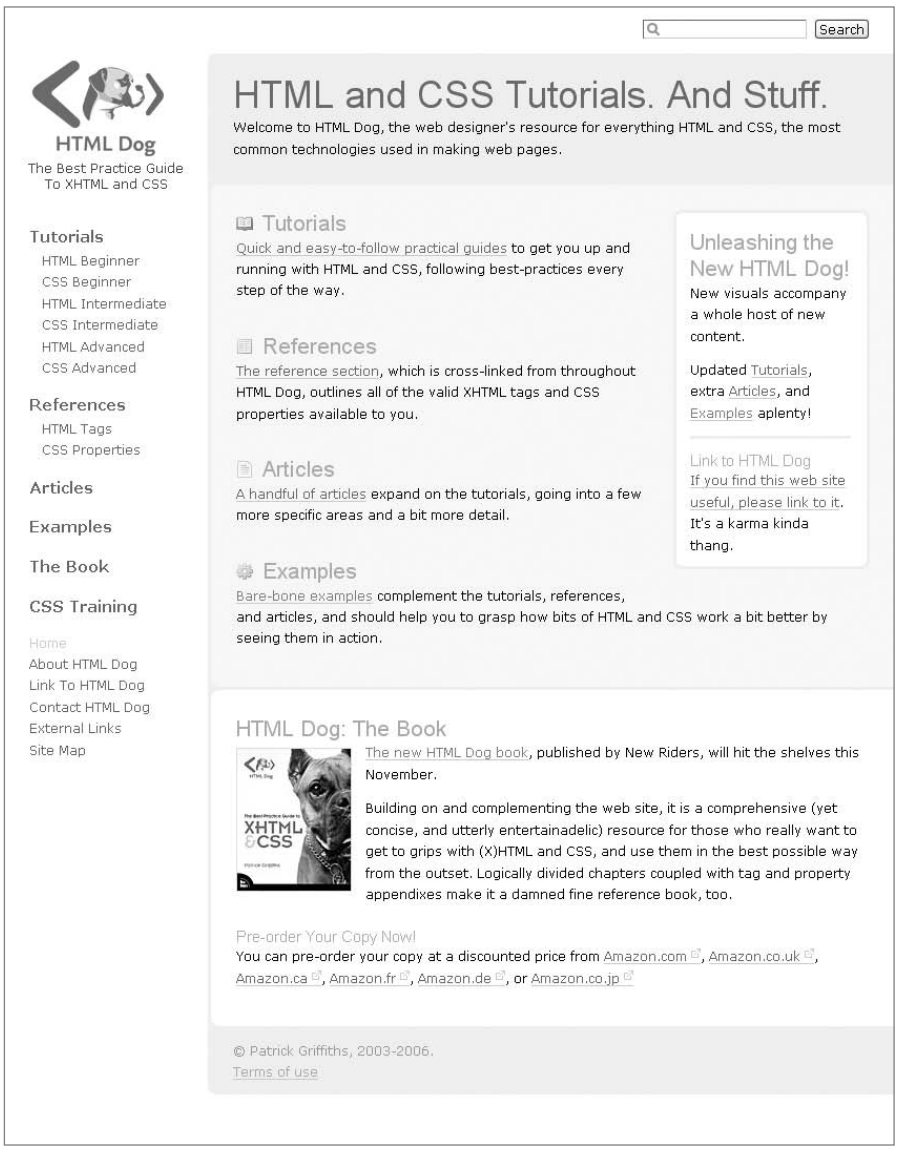

Search
HTML Dog
The Best Practice Guide To XHTML and CSS
Tutorials
HTML Beginner
CSS Beginner
HTML Intermediate
CSS Intermediate
HTML Advanced
CSS Advanced
References
HTML Tags
CSS Properties
Articles
Examples
The Book
CSS Training
Home
About HTML Dog
Link To HTML Dog
Contact HTML Dog
External Links
Site Map
HTML and CSS Tutorials. And Stuff.
Welcome to HTML Dog, the web designer's resource for everything HTML and CSS, the most common technologies used in making web pages.
Tutorials
Quick and easy-to-follow practical guides to get you up and running with HTML and CSS, following best-practices every step of the way.
References
The reference section, which is cross-linked from throughout HTML Dog, outlines all of the valid XHTML tags and CSS properties available to you.
Articles
A handful of articles expand on the tutorials, going into a few more specific areas and a bit more detail.
Examples
Bare-bone examples complement the tutorials, references, and articles, and should help you to grasp how bits of HTML and CSS work a bit better by seeing them in action.
Unleashing the New HTML Dog!
New visuals accompany a whole host of new content.
Updated Tutorials, extra Articles, and Examples aplenty!
Link to HTML Dog
If you find this web site useful, please link to it. It's a karma kinda thang.
HTML Dog: The Book
XHTML & CSS
The new HTML Dog book, published by New Riders, will hit the shelves this November.
Building on and complementing the web site, it is a comprehensive (yet concise, and utterly entertainadelic) resource for those who really want to get to grips with (X)HTML and CSS, and use them in the best possible way from the outset. Logically divided chapters coupled with tag and property appendixes make it a damned fine reference book, too.
Pre-order Your Copy Now!
You can pre-order your copy at a discounted price from Amazon.com, Amazon.co.uk, Amazon.ca, Amazon.fr, Amazon.de, or Amazon.co.jp
© Patrick Griffiths, 2003-2006.
Terms of use

FIGURE 0.1 This book expands on the popular HTML Dog website: *http://www.htmldog.com*.

So What Are XHTML and CSS?

XHTML (eXtensible HTML) is the latest, Sly-Stone-funkiest version of HyperText Markup Language. HTML is a simple language used to structure hyperlinking content that is at the core of most web pages. The whole idea behind HTML since Day One has simply been to apply meaning to chunks of content and link them all together, regardless of platform.

In technical terms, XHTML is "HTML reformed as XML," but to most people, and for the purpose of this book, it simply means a more modern version of HTML (which is why "HTML" is usually referred to throughout the book rather than "XHTML") with an ever so slightly stricter syntax (explained in Chapter 1, "Getting Started"). Although there are lesser, "transitional" versions of XHTML, this book jumps in at the deep end and follows XHTML Strict—the purest form of XHTML, which harnesses all of its intended benefits. If this sounds daunting to you, don't worry—it's really not more difficult to use than other versions, but it is the best one to opt for if you have the ability to do so.

CSS (Cascading Style Sheets) is a language used to present a web page that was introduced to remedy the increasing introduction (by browser manufacturers) and use of presentational HTML elements. Not only does it lend even greater control over the appearance of a web page, it removes the need for presentational elements in the HTML document itself. It has taken a while for browsers to cotton on, but most CSS is now supported by a vast majority of browsers. Its use is not only a genuine option, but also the best option for presenting web pages. As you will see later in this Introduction, this ability to separate content (HTML) and presentation (CSS) leads to great benefits. What are style sheets and how do they cascade? See Chapter 1.

As with HTML, there are various versions of CSS. This book largely follows CSS 2.1, the complete and widely supported revised version of CSS 2. This provides control not only over basic font decorations, but powerful positioning capabilities (goodbye, table layouts!) and the handling of different media types.

What Are Web Standards?

Web standards are universal rules that dictate how something should be used, independent of any single thing (such as one particular browser). By utilizing web standards you are helping to ensure universal compatibility and flexibility. Because they are based on logical reasoning with no commercial pressure (well, that's the idea anyway!), following them also tends to lead to greatly optimized solutions.

These standards are the creation of the W3C (World Wide Web Consortium, *www.w3.org*), an independent body that counts Google, Intel, AOL, Apple, various universities, the BBC, Sony, Microsoft, and many more amongst its members and is contributed to by hundreds in the web community. The standards are wide-ranging, encompassing a large number of web technologies and initiatives, including HTML and CSS.

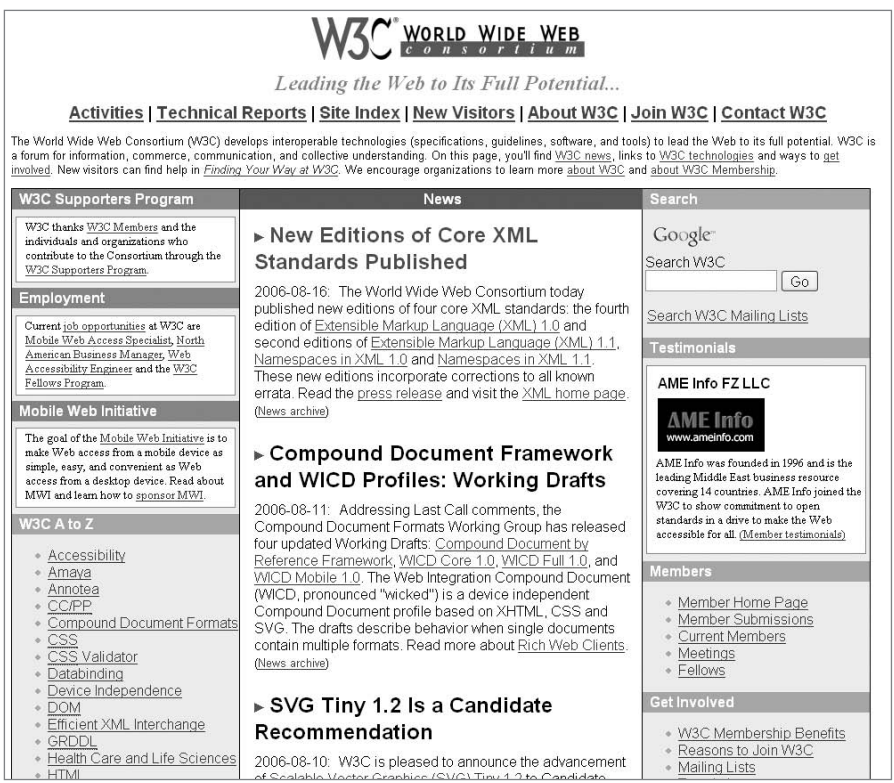

W3C WORLD WIDE WEB consortium
Leading the Web to Its Full Potential...
Activities | Technical Reports | Site Index | New Visitors | About W3C | Join W3C | Contact W3C
The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding. On this page, you'll find W3C news, links to W3C technologies and ways to get involved. New visitors can find help in Finding Your Way at W3C. We encourage organizations to learn more about W3C and about W3C Membership.
W3C Supporters Program
W3C thanks W3C Members and the individuals and organizations who contribute to the Consortium through the W3C Supporters Program.
Employment
Current job opportunities at W3C are Mobile Web Access Specialist, North American Business Manager, Web Accessibility Engineer and the W3C Fellows Program.
Mobile Web Initiative
The goal of the Mobile Web Initiative is to make Web access from a mobile device as simple, easy, and convenient as Web access from a desktop device. Read about MWI and learn how to sponsor MWI.
W3C A to Z
Accessibility
Amaya
Annotea
CC/PP
Compound Document Formats
CSS
CSS Validator
Databinding
Device Independence
DOM
Efficient XML Interchange
GRDDL
Health Care and Life Sciences
HTML
News
New Editions of Core XML Standards Published
2006-08-16: The World Wide Web Consortium today published new editions of four core XML standards: the fourth edition of Extensible Markup Language (XML) 1.0 and second editions of Extensible Markup Language (XML) 1.1, Namespaces in XML 1.0 and Namespaces in XML 1.1. These new editions incorporate corrections to all known errata. Read the press release and visit the XML home page. (News archive)
Compound Document Framework and WICD Profiles: Working Drafts
2006-08-11: Addressing Last Call comments, the Compound Document Formats Working Group has released four updated Working Drafts: Compound Document by Reference Framework, WICD Core 1.0, WICD Full 1.0, and WICD Mobile 1.0. The Web Integration Compound Document (WICD, pronounced "wicked") is a device independent Compound Document profile based on XHTML, CSS and SVG. The drafts describe behavior when single documents contain multiple formats. Read more about Rich Web Clients. (News archive)
SVG Tiny 1.2 Is a Candidate Recommendation
2006-08-10: W3C is pleased to announce the advancement
Search
Google
Search W3C
Go
Search W3C Mailing Lists
Testimonials
AME Info FZ LLC
AME Info
www.ameinfo.com
AME Info was founded in 1996 and is the leading Middle East business resource covering 14 countries. AME Info joined the W3C to show commitment to open standards in a drive to make the Web accessible for all. (Member testimonials)
Members
Member Home Page
Member Submissions
Current Members
Meetings
Fellows
Get Involved
W3C Membership Benefits
Reasons to Join W3C
Mailing Lists

FIGURE 0.2 The W3C's site (*www.w3.org*), although a little difficult to penetrate, is a great source of information.

HTML standards are based on semantics. This is the process of using tags to apply meaning, such as "this piece of text is a paragraph" or "the HTML in this element makes a table." It may not sound that important, but it's at the heart of the first step toward web standards success: separating structure and presentation.

Structure and Presentation

This page, and in particular the big fat statement below, is more important than the rest of this book put together.

HTML = CONTENT
CSS = PRESENTATION

This is the magical key to unlocking better web pages. If you apply this rule when building web pages then you're already halfway toward achieving web standards and the benefits that come with them.

A central philosophy of web standards is "separation of content and presentation." Variations include "separation of meaning and presentation" and "separation of structure and presentation." There are pedantic arguments why one of these is better than another, but all of the phrases are perfectly valid and essentially imply the same thing. If we were more precise (but less snappy), we might say "the separation of [content made meaningful by structure] and [presentation]." What it means is that HTML is for one thing and CSS is for the other—HTML should be used solely to construct content and CSS should be used solely to present it.

Amongst other things, this means that `font` tags are out and it means tables are fine for tabular data, but not for layout (see Chapter 8, "Tables"). It also means that when you have a heading it should be marked up as a heading, not just presented to look like one, and it means that `em` tags should be used for emphasis, not just CSS decoration.

FIGURE 0.3 The CSS Zen Garden—a great example of separating content and presentation. Design by Didier Hilhorst.

CSS ZEN GARDEN
THE BEAUTY OF CSS DESIGN
Select a design
Archives
Resources
Why participate?
We would like to see as much CSS as possible.
Unfortunately, designing this way highlights the
This is a learning exercise as well as a
Bandwidth graciously donated by DreamFlow
Graphic artists only please.
You may modify the style sheet in any way you
Download the sample html file

FIGURE 0.4 With exactly the same XHTML, different CSS can achieve radically different designs. Design by Samuel Marin.

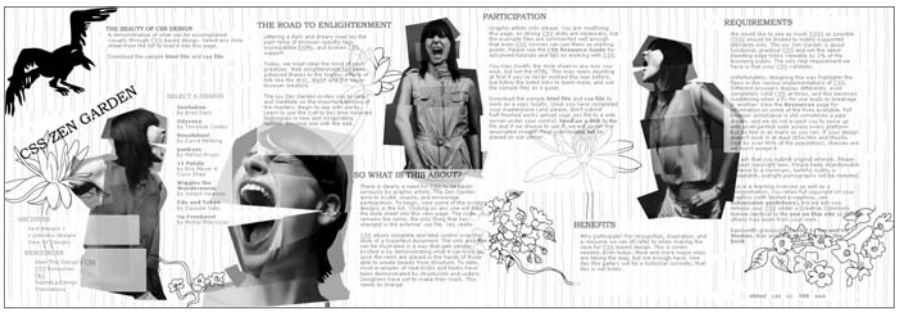


FIGURE 0.5 The structured content, independent of presentation, is all in place as it should be, and it can be presented however the designer chooses. Design by Mikhel Proulx.

If there's anything you're doing with HTML that even hints at presentation, *stop*—CSS can do it better. That means everything from using it just to italicize text to using tables for layout. And if there's content that has genuine specific meaning (such as emphasis), making it look different with CSS isn't enough—HTML should be used to apply that meaning through structure.

Example	Structure or Presentation?	HTML or CSS?
A heading	Structure	XHTML
Size of a heading	Presentation	CSS
A paragraph	Structure	XHTML
Color of the text in a paragraph	Presentation	CSS
A table of figures	Structure	XHTML
A border around table cells	Presentation	CSS
An image, such as a portrait photo	Structure	XHTML
An image, such as a tessellating background	Presentation	CSS
A group of navigation links	Structure	XHTML
The placement of a group of navigation links on a page	Presentation	CSS

This isn't just about whether valid tags are being used individually, because a page where tables are used for layout, for example, or where `b` (for bold) tags are being used can quite plausibly be a "valid" page. It's about the bigger picture and good

practice through using HTML tags as they were intended—to mark up data and apply meaning to it.

The majority of the practical benefits of using web standards stem from this philosophy. It's a good one. It works.

The Practicalities of Web Standards

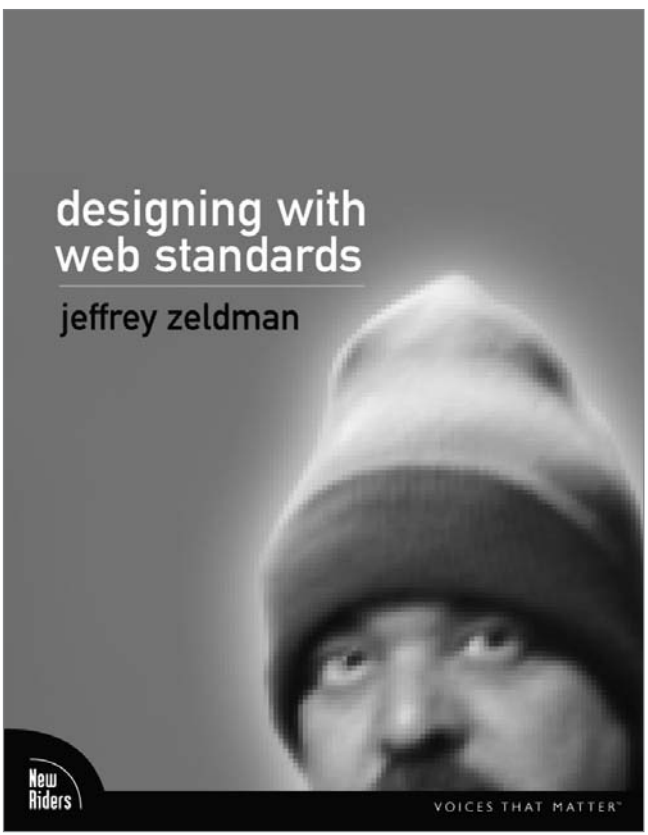


FIGURE 0.6 Jeffrey Zeldman's *Designing With Web Standards* set the scene by explaining what web standards are all about and why we should be using them.

Web standards aren't just some form of Jedi mind trick, and web standards evangelists tend not (too often) to be loony cult members. No one would seriously take any notice of web standards, let alone use them, if there weren't real, substantial, practical benefits. Here are some of the most important ones:

- **Cross-compatibility:** What sets web standards apart from the old school is that there's nothing specific to one browser or another. A common web design problem has been the "necessity" of developing one page for one browser and one page for another. Even today, some sites are made to work solely in Microsoft's Internet Explorer (IE), excluding a small, but significant, percentage of users. But barring a few minor discrepancies, following web standards will ensure that pages will work everywhere. No alternative versions needed, no "IE only (we're too lazy)," no exclusion of users.
- **Forward compatibility:** Thanks to the increased likelihood of cross-compatibility, web pages will be more likely to work as desired on future browsers than they would be if they depended on the nonstandard proprietary oddities of current browsers.
- **Centralized control of presentation:** As will be explained in Chapter 1, following web standards and using CSS allows a single, or multiple, global file(s) to apply presentation to all web pages across a site. Separating out the presentation from the HTML in this way makes it much easier and quicker to make site-wide presentational changes, resulting in a more consistent design than a situation in which you have to change every web page individually.
- **Device independence:** Think that everyone looking at a web page is staring at a computer screen? Well, most of them are, but some might be printing out web pages or using some form of mobile device (Chapter 10, "Multiple Media"). You might think that to properly accommodate multiple devices you would need multiple versions of web pages. But you don't, not with web standards. By separating structure and presentation, the same content can be displayed differently depending on the device.

- **Search engine optimization:** Search engines love web standards. They used to love metadata (data about the web page explicitly written into it by the author) but this subjective tagging was easy to exploit and led to search results that weren't necessarily that relevant. Now search engines are much more sophisticated and use more advanced techniques to rate the relevancy of a page to a search query. They tend to analyze the content itself and take special interest in things such as headings and even how close relevant content is to the top of the page. So if you're using `font` tags to make text look like headings instead of `h1`s and `h2`s (see Chapter 2, "Text") or if you've got all of that table mess surrounding more table mess surrounding the content, then you're not doing yourself any favors.

- **Lightweight pages:** Perhaps the most immediately impressive advantage is just how lightweight pages become. Lighter pages mean decreases in bandwidth (reducing hosting costs) and web page loading time (increasing usability). An equivalent "old-school" page made with tables for layout and `font` tags for text decoration is a relatively fat load of markup. Without the need for such bulky code or unnecessary graphics (such as transparent "spacer" images and graphical text that could be replicated with CSS), it isn't uncommon to produce pages that are as much as 60 or 70 percent lighter.

- **Accessibility:** Making it easier for users with disabilities to access web pages satisfies a moral duty, opens up your website to a wider market, and helps it to comply with antidiscrimination laws. A large proportion of web accessibility issues are technical and are tackled by using good web-standard HTML and CSS.

- **Employer and client expectations:** If you're not sold on web standards, plenty of other people are. The ability to code to W3C (X)HTML, CSS, and accessibility standards is fast becoming a must-have skill in a web designer. If you are on the market for a web design job or if you design sites for others, having web standards in your arsenal is massive plus.

BROWSER COMPATIBILITY ISSUES

A fundamental principle behind web standards is that they are browser-independent. You shouldn't have to create browser-specific code—the whole idea is that if browser manufacturers fully supported the web standards laid out by the W3C, then one page could suit all. But we live in the real world and no browser is perfect. When you think of all the technical intricacies—their syntax and their behavior—involved in HTML and CSS, it's little surprise that not every rule is applied 100 percent correctly, if applied at all, by every browser.

Does this mean web standards are useless in practical terms? No. The great news is that all popular modern browsers support a vast majority of web standards. It's just a few little niggles that sometimes cause irritation. A little scratch and they tend to go away.

By far the most popular browser out there is Internet Explorer for Windows. IE is pretty much universally derided in the web standards community because it has many shortcomings—either lack of support or incorrect interpretations of quite a few web standard rules.

More modern browsers, such as Firefox and Safari, are technically superior pieces of software (a statement that few could truthfully argue against—even Microsoft), but unfortunately for the web designer (and to the ultimate detriment of the user) there are only a handful of computer users who use anything other than the pre-installed browser on their machine. This means predominantly Windows and this means Internet Explorer.

But IE is not a complete idiot. It handles most areas of HTML and CSS W3C standards very well. There are no gaping holes that prevent an author from achieving a certain page structure or force him or her to compromise on a particular layout, for example. It's only when it comes down to more specific details that incompatibilities can get frustrating. The good news is that Microsoft has finally gotten the message and has worked hard to fix many of these problems in the latest, seventh incarnation of IE, which has now landed on Planet Web.

Browser shortcomings will be pointed out where applicable throughout this book, but in general terms there is absolutely no practical reason not to adopt web standards—all modern browsers are more than capable to deal with them.

We do have to work in a multibrowser world and even the best web makers encounter discrepancies in their pages between browsers. They are usually easy to iron out, and as long as you test your pages on multiple browsers to make sure designs work, compatibility issues shouldn't cause too many headaches.

BROWSER HACKS

It isn't true that lots of "hacks" are needed for web standard design. In fact, that's a bit of a contradiction.

There is a large quantity of hacks out there (particularly for CSS) that allow you to dish out different code to different browsers, but they are generally just unnecessary quick fixes for something that wasn't constructed properly in the first place. In fact, in practical terms, there isn't usually a need to use anything other than one simple hack—the box model hack—that is necessary to accommodate a calculating error in IE 5.x (see Chapter 5, "Layout").

THE DANGERS OF BACKWARDS COMPATIBILITY

Working on data collected from the Middle Ages, there are still those who bang on about Netscape 4. "Does it work on Netscape 4? Because Netscape 4's really important. It has to work in Netscape 4."

It's the most infamous of the "backwards compatibility" arguments, but it's also the best example of taking backwards compatibility too far.

The first step on judging whether you should accommodate a browser is the number of people who use it. The second step is judging to what extent you have to compromise a web design to accommodate that browser.

Only a tiny fraction of a percentage of people now use Netscape 4. But even though it could be accommodated (a tiny percentage is still a percentage, after all), it's not worth it.

A confused sage once said "The pinnacle of good web design is a web design that works on all browsers."

Piffle.

Yes, a web design should work on as many browsers as possible, but at what cost?

Bending over backwards to accommodate old browsers will be to the detriment of those who use newer browsers. What you lose by accommodating old browsers are the practical benefits of web standards, mentioned above. Wave goodbye to flexibility, lighter pages, increased accessibility, heightened usability, and lower maintenance. You are going to lose more visitors through lack of optimization than you are going to gain through the accommodation of obscure antiques.

The content of a well-structured HTML document should still be completely accessible on older browsers—those that do not understand CSS or those that are tricked into ignoring it (see "Applying CSS" in Chapter 1) will simply render the HTML in the browser's default style. The design may be lost, but the functionality won't be.

ACCESSIBILITY

Accessibility is all about how easy it is for someone, in whatever situation, to gain access to something. Although this is quite a general area (and in relation to the Web it can include access to a site via "alternative" devices such as mobiles, for example), it tends to focus on issues that arise concerning people

with disabilities and how easy it is for them to access the information on a web page or website.

The extent of accessibility considerations can fill a whole book (and they do—see Joe Clark's excellent *Building Accessible Websites* (New Riders) or *Web Accessibility: Web Standards and Regulatory Compliance* (Friends of Ed), for example—but we will explore them a bit here because HTML and CSS are the tools ultimately used to tackle a majority of accessibility issues.

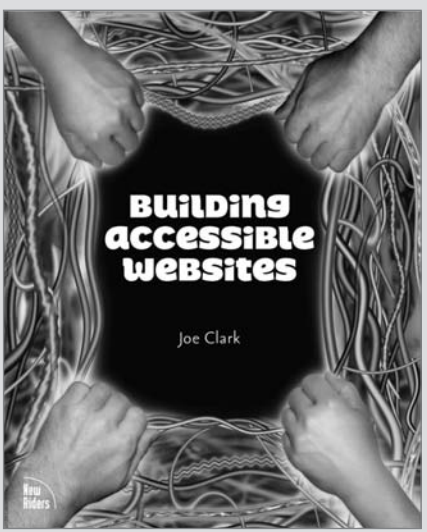


FIGURE 0.7 Although some years old, *Constructing Accessible Websites* by Joe Clark is still the best, most comprehensive source for understanding web accessibility and applying accessibility techniques in web design.

In many cases, very simple HTML and (to a lesser degree) CSS steps can greatly improve accessibility, but there are a lot of those simple steps, particularly with components of a page that require interaction—see Chapter 3, "Links," and Chapter 9, "Forms." Although the issues surrounding various accessibility initiatives are not explicitly explored in this book (again, that is something for another book), the techniques for achieving most of them are.

Who Is This Book For?

This book is for those who want to get to grips with best-practice (X)HTML and CSS, and for those who want a solid, reliable reference book.

Although the topic of web standards may appeal more to intermediate-to-advanced web designers, the comprehensive nature of the book should suit beginners and experts alike, both as a guide through how to author components of a web page and as a reference to make sure you're using the correct syntax.

I've written something that I myself would find useful now, but something I also would have found very useful back when I was first getting to grips with HTML, CSS, and web standards.

How This Book Works

The book comprehensively works through the various components of a web page (except Chapter 10, which is slightly different), explaining how to structure them and how to present them. Component by component, by the end of the book, all practical web standards (XHTML 1.0 Strict and CSS 2.1) tools will have been covered.

Practical Web Standards

Due to the current state of browser compatibility, not every W3C detail is covered in this book because even as we promote the philosophy of using web standards, we must also be practical. There's no point in banging on about a technique that is fine in theory but doesn't work in a majority of browsers. It would be a waste of paper and a waste of your time. Be secure in the knowledge that most web standards options are practical and are covered. It's just a baby's handful of pesky goblins that try to spoil the fun.

Note that Appendixes A and B cover every valid (nonpresentational—see below) HTML (XHTML 1.0 Strict) tag, CSS (2.1) property, and every valid attribute and value. When browser incompatibilities crop up, a note to that effect will be attached.

In the name of good practice, presentational HTML tags such as `b` and `i` (that are actually valid XHTML 1 tags) are also banished. We're going with the separation of

structure and presentation here and the practical benefits it brings, so there's no room for these dated lingerers that are destined for the scrap heap anyway.

A brief look at some of the commonly used tags and attributes that don't fit into the philosophy of this book (mainly invalid tags, but also tags such as `b` and `i`) are noted in the "Bad Tags" section of Appendix A.

www.htmldog.com

There should be enough in this book to make at least a small cluster of your brain cells feel all warm and fuzzy, but part of its design is to work hand-in-hand with the HTML Dog website to give you even more help with HTML and CSS. Throughout the book, you will find references to articles, which might go into more detail about certain techniques, for example, and you will also find numerous pointers to "bare-bone" examples. These examples were designed to strip away all but the necessary code to demonstrate a small part of HTML or CSS, such as headings, or forms, background images, or vertical alignment. Simply view the page source (an option which can be found under the "View" menu item of most browsers) to see what's going on.

When they pop up in the book, they'll look a little something like this:

 www.htmldog.com/examples/verticalalign.html

A list of the gamut of 70-odd examples can be found at *www.htmldog.com/examples/*

The Chapters

Neatly nestled into 10 chapters you will find explanations of pretty much every component of HTML and CSS you'll need, along with a few fancy techniques to add a little bit of pizzazz to your pages.

- **Chapter 1, "Getting Started"**—sets the ball rolling by explaining the syntax of HTML and CSS: what they look like, how they should be used, and how they can be linked together.
- **Chapter 2, "Text"**—covers all of the HTML tags used to structure various types of text: paragraphs, headings, emphasis, abbreviations, and much

more. The chapter then looks at the CSS that can be used to apply things such as fonts, sizes, italics, and character spacing.

- **Chapter 3, "Links"**—looks at just one tag, but it's such an important one that it has been honored with its own chapter. From basic links and page anchors through to making links more accessible and good practices in styling them.
- **Chapter 4, "Images"**—covers how to add content with the `img` tag and how to add striking presentation with the powerful CSS background image.
- **Chapter 5, "Layout"**—explains how you can achieve various layouts using CSS.
- **Chapter 6, "Lists"**—goes over ordered, unordered, and definition lists and how they can be styled to make page components such as navigational tabs.
- **Chapter 7, "Scripts & Objects"**—explains how JavaScript and objects such as Flash movies can be incorporated into an HTML page.
- **Chapter 8, "Tables"**—covers everything you need to know about how to mark up tabular data (not how to use tables for layout!), including how to make tables more accessible. There are also a few specific CSS techniques thrown in that can be used to make their presentation all the prettier.
- **Chapter 9, "Forms"**—covers how to structure and present forms and form fields for user input, and explains the limitations of styling form elements.
- **Chapter 10, "Multiple Media"**—looks at how web pages work in media other than your standard desktop or laptop computer, and how you can optimize the CSS of your web pages (without touching the HTML) so they are displayed more appropriately when printed out.

chapter 1

Getting Started

GETTING STARTED is often the most difficult thing to do. Sometimes it's easier not to start at the beginning, but rather just jump in halfway and start messing about with images or forms, for example. That might get you on the road to a more interesting-looking web page quicker, but your car would probably be in better condition afterwards if you learned how to drive properly first. It's the same as with any other subject—there's always a whole load of theory to plow through, but getting through it will make life easier and better in the long run.

This chapter will tell you pretty much everything you need to know about putting together the basic components of a web page. It splits quite neatly into two—how to use HTML and how to use CSS.

HTML Syntax

HTML has a very straightforward syntax: Content is structured into elements using tags with extra information supplied by attributes. XHTML, which we'll be using, has a stricter syntax than older (non-X) HTML versions, but if you follow the simple rules, you should reap the benefits.

Elements, Tags, and Attributes

All you are doing with HTML is taking content and defining what each piece of it means by wrapping the pieces in tags. To define a few terms, in the following example:

```
<a href="http://www.htmldog.com/">HTML Dog</a>
```

"`<a href="http://www.htmldog.com/">`" is an opening tag, which defines the start of an element.

"`</a>`" is a closing tag, which defines the end of an element.

"`href`" is an attribute, which is a setting for an element. (In this example, "`href`" is the destination of a link—see Chapter 3, "Links.")

"`http://www.htmldog.com/`" is an attribute value, used to specify what the attribute should be set to.

"`HTML Dog`" is content.

"`<a href="http://www.htmldog.com/">HTML Dog</a>`" (the whole shebang) is an element.

There are a few simple rules to follow when it comes to tags (besides using valid tags and attributes, which the chapters will cover).

Firstly, XHTML requires that **all tags and attributes must be lowercase**. `<p></p>` and `<blockquote></blockquote>` are valid, but `<P></P>` and `<BLOCKQUOTE></BLOCKQUOTE>` are not. (If you aren't familiar with some of the tags in this chapter, don't worry; they are covered later in the book.)

Secondly, **all tags must close**. In the above example, the end of the element is marked by `</a>`. `<h1>` must be closed with an `</h1>`, `<div>` must be closed with a `</div>`, and so on. There are special cases where an element has no content, such as `br` or `input`. In these cases there is no explicit closing tag, but rather the single tag closes itself with the "/" character at the end, as in `<br />` or `<input />`.

Thirdly, **all attribute values must be in quotation marks** (and all attributes must have values). For example, `<a href=http://www.htmldog.com>HTML Dog</a>` is not valid—it must be `<a href="http://www.htmldog.com">HTML Dog</a>`.

Fourthly, **elements must be nested properly**.

Nested elements are elements enclosed in other elements.

An example is:

```
<p>Why not try out <a href="http://www.htmldog.com/">HTML Dog</a>?</p>
```

In this case, the `a` element (a link—see Chapter 3) is nested inside the `p` element (a paragraph—see Chapter 2, "Text").

You have to be careful when nesting elements—one must fit snugly inside another. So, for example,

```
<em><a href="http://www.htmldog.com/">HTML Dog</a></em>
```

is good, but

```
<em><a href="http://www.htmldog.com/">HTML Dog</em></a>
```

is not. If the `a` element is to be inside the `em` element (emphasis—see Chapter 2) then the closing tag for the `a` element must come before the closing `em` tag.

IT'S A FAMILY AFFAIR

The relationship of one element to another can be defined in terms of family connections. With nested elements, an element within another element can be called a *child* of the containing element. In turn, the containing element is known as the *parent* of that child.

So in `<p><em>Lemon</em> pie</p>`, the `p` element is the parent of the `em` element, which is the child of the `p` element.

You will also come across terms such as *siblings, ancestors*, and *descendants*.

BLOCK AND INLINE ELEMENTS

All HTML elements are one of two types—block or inline.

Block elements collect together other block elements or inline elements, or even plain old textual content, and are used to structure something that is greater than a simple line of content. They include `div` (used to divide up code by splitting it into chunks—explained in detail later), `p` (paragraphs—see Chapter 2) and `table` (Chapter 8, "Tables").

Inline elements are just that—elements within a line. They include `span` (see later), `em` (emphasis—see Chapter 2) and `img` (image—see Chapter 4, "Images").

Keep in mind that you can't have a block element inside an inline element (such as `<em><p>Ra ra</p></em>`). See Appendix A for more details on what elements can be nested within certain elements.

Common Attributes

Throughout this book you will come across many attributes that are specific to certain tags or collections of tags. There is, however, a group of "common attributes" that can be used with most tags.

The common attributes consist of core, i18n, and event attributes.

Core attributes

The core attributes are `class`, `id`, `title`, and `style`.

Classes and ids apply an extra little label to an element, and are used for page anchors (a position on a page to which a link can jump, as explained in Chapter 3), manipulation of elements with JavaScript, and, most commonly, as a way of directly targeting an element with CSS.

```
<div id="content">
    <p class="chair">Lorem ... ipsum ... etc.</p>
    <p>Lorem ... schmipsum ... etc.</p>
    <p class="chair">Etc. ... ipsum ... schmipsum.</p>
</div>
```

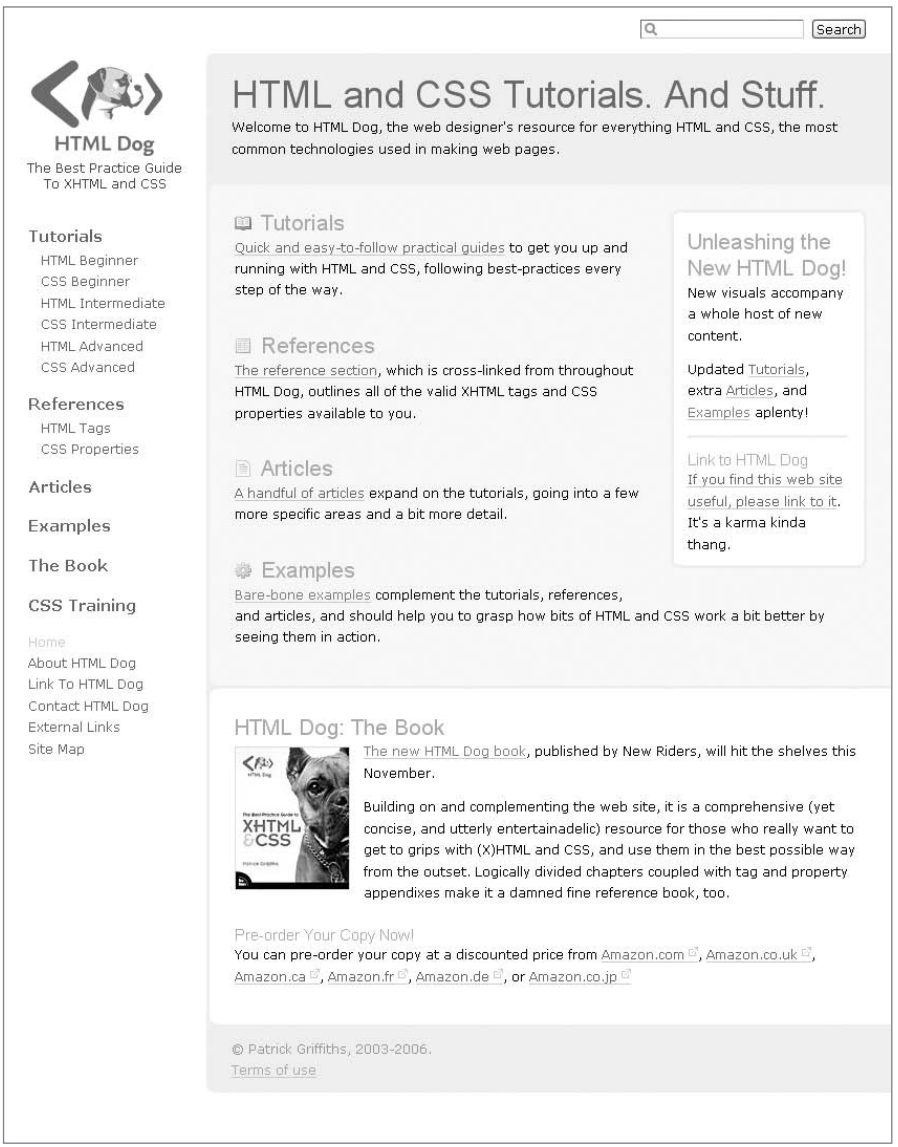

Search
HTML Dog
The Best Practice Guide To XHTML and CSS
Tutorials
HTML Beginner
CSS Beginner
HTML Intermediate
CSS Intermediate
HTML Advanced
CSS Advanced
References
HTML Tags
CSS Properties
Articles
Examples
The Book
CSS Training
Home
About HTML Dog
Link To HTML Dog
Contact HTML Dog
External Links
Site Map
HTML and CSS Tutorials. And Stuff.
Welcome to HTML Dog, the web designer's resource for everything HTML and CSS, the most common technologies used in making web pages.
Tutorials
Quick and easy-to-follow practical guides to get you up and running with HTML and CSS, following best-practices every step of the way.
References
The reference section, which is cross-linked from throughout HTML Dog, outlines all of the valid XHTML tags and CSS properties available to you.
Articles
A handful of articles expand on the tutorials, going into a few more specific areas and a bit more detail.
Examples
Bare-bone examples complement the tutorials, references, and articles, and should help you to grasp how bits of HTML and CSS work a bit better by seeing them in action.
Unleashing the New HTML Dog!
New visuals accompany a whole host of new content.
Updated Tutorials, extra Articles, and Examples aplenty!
Link to HTML Dog
If you find this web site useful, please link to it. It's a karma kinda thang.
HTML Dog: The Book
XHTML &CSS
The new HTML Dog book, published by New Riders, will hit the shelves this November.
Building on and complementing the web site, it is a comprehensive (yet concise, and utterly entertainadelic) resource for those who really want to get to grips with (X)HTML and CSS, and use them in the best possible way from the outset. Logically divided chapters coupled with tag and property appendixes make it a damned fine reference book, too.
Pre-order Your Copy Now!
You can pre-order your copy at a discounted price from Amazon.com, Amazon.co.uk, Amazon.ca, Amazon.fr, Amazon.de, or Amazon.co.jp
© Patrick Griffiths, 2003-2006.
Terms of use

FIGURE 1.1 The illustrations in this chapter are taken from the HTML Dog website (*www.htmldog.com*).

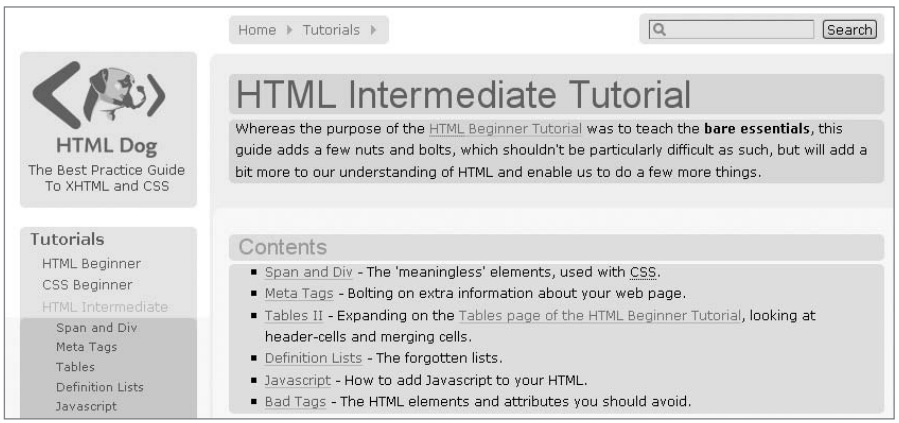

Home ▸ Tutorials ▸
Search
HTML Dog
The Best Practice Guide To XHTML and CSS
HTML Intermediate Tutorial
Whereas the purpose of the HTML Beginner Tutorial was to teach the bare essentials, this guide adds a few nuts and bolts, which shouldn't be particularly difficult as such, but will add a bit more to our understanding of HTML and enable us to do a few more things.
Tutorials
HTML Beginner
CSS Beginner
HTML Intermediate
Span and Div
Meta Tags
Tables
Definition Lists
Javascript
Contents
Span and Div - The 'meaningless' elements, used with CSS.
Meta Tags - Bolting on extra information about your web page.
Tables II - Expanding on the Tables page of the HTML Beginner Tutorial, looking at header-cells and merging cells.
Definition Lists - The forgotten lists.
Javascript - How to add Javascript to your HTML.
Bad Tags - The HTML elements and attributes you should avoid.

FIGURE 1.2 A few examples of the components that are block elements: paragraphs, headings, forms, and lists. The list items are also block elements.

`id`s are used when there is just one unique element that needs a CSS association (or an anchor) and uniquely identifies a part of a document (such as "content" in the above example). Only one element in an HTML document can have an id with a certain value so for example, you can't have:

```
<h2 id="plant">Tree</h2>
<h2 id="plant">Bush</h2>
```

Unlike ids, any number of elements in an HTML document can have a class with a certain value. They are used when there is more than one element that needs the same CSS association, so, for example, you could have:

```
<h2 class="plant">Tree</h2>
<h2 class="plant">Bush</h2>
```

Classes and ids will come up again in this chapter, when we look at class and id CSS selectors.

`title` adds a title to an element. A handy little critter, `title` can be used to add a bit more information. This is commonly used with elements such as `abbr` to define the phrase that an abbreviation is representing (see Chapter 2); `blockquote`, to give more information on where a quote has come from (again, see Chapter 2); or `a`, to give more information on what to expect at the destination of a link (see Chapter 3).

The value of a `title` attribute can be read out by screen readers (increasing accessibility), and browsers will commonly turn the value of the `title` attribute into a little "tool tip," popping it up by the cursor when it moves over the element. This can be useful in providing more information about a certain element, such as what an acronym stands for or where a link will take the user.

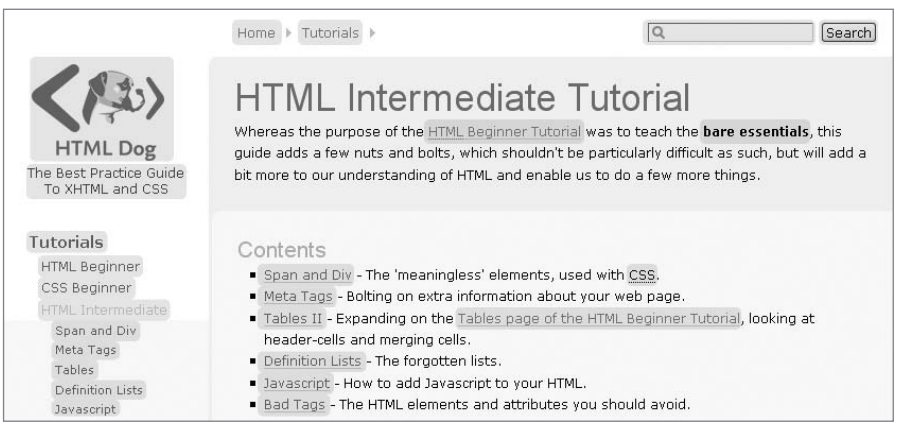

Home
Tutorials
Search
HTML Dog
The Best Practice Guide To XHTML and CSS
HTML Intermediate Tutorial
Whereas the purpose of the HTML Beginner Tutorial was to teach the bare essentials, this guide adds a few nuts and bolts, which shouldn't be particularly difficult as such, but will add a bit more to our understanding of HTML and enable us to do a few more things.
Tutorials
HTML Beginner
CSS Beginner
HTML Intermediate
Span and Div
Meta Tags
Tables
Definition Lists
Javascript
Contents
Span and Div - The 'meaningless' elements, used with CSS.
Meta Tags - Bolting on extra information about your web page.
Tables II - Expanding on the Tables page of the HTML Beginner Tutorial, looking at header-cells and merging cells.
Definition Lists - The forgotten lists.
Javascript - How to add Javascript to your HTML.
Bad Tags - The HTML elements and attributes you should avoid.

FIGURE 1.3 ...And a few examples of inline elements: links, form fields, images, and emphasized text.

The `style` attribute, which is used to inject CSS directly into the HTML (with a blunt, uncomfortable needle), will be explained later (as will the reference to the blunt, uncomfortable needle) under the "Applying CSS to HTML" heading.

i18n attributes

The i18n attributes, so called because few people can be bothered to write the 18 characters in between *i* and *n* in *internationalization,* are `dir` and `xml:lang`.

`dir` specifies the direction of content. Values can be `ltr` (left to right—for languages such as English) or `rtl` (right to left—for languages such as Arabic).

`xml:lang` specifies the language of the content of an element, such as `en` for English, `de` for German or `mg` for Malagasy.

Event attributes

The `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, and `onkeyup` attributes invoke the JavaScript value when the user takes certain actions. You can read more about event attributes, and why you should avoid using them, in Chapter 7, "Scripts & Objects."

The Basic Structure of an HTML Document

A number of basic structural elements are required to make a valid (X)HTML page.

Basically, everything should fit into a structure outline that looks something like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>
    <title></title>
</head>

<body>
</body>

</html>
```

At the very top is a document-type declaration and following that is an `html` element. Inside the `html` element there are two elements—`head` and `body`. The contents of the `head` element (including the required `title` element) give general information about the content of the HTML document. The content of the `body` element is where everything else goes—the viewable (or audible, or otherwise experienced) web page content.

Declarations

There are a few things that need to be done to define a valid HTML document before really getting stuck in to the HTML. A document-type declaration lets the browser know what version of HTML you're using, the primary language should also be stated, and you also need to specify the file type and character set of the docu-

ment. This might sound a bit daunting, but all it involves is a few lines of standard code at the top of your web page. Once the code's there you don't have to worry about it.

Document Type At the very top of your web pages, you need a document declaration. That's right, you *need* it.

Without specifying a doctype, your HTML just isn't valid HTML and most browsers displaying it will switch to "quirks mode," which means they will assume that you, the author, don't have a clue what you're doing and so they will make up their own mind about what to do with your code.

At this moment in time, the best document declaration to use in most situations is for XHTML 1.0 Strict. And it looks like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

The following is the document declaration for XHTML 1.1, which may seem preferable, being the latest version of XHTML, but there are a few problems with browser compatibility (because a lot of them don't really know about it yet). To the web page author, this has few differences from XHTML 1.0 Strict anyway.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

This line of code (which is usually broken in two, as above, just to make things a little neater) tells the browser what version of HTML to expect and where the "document type definition" can be found. It must be at the very top of the HTML document, with no content preceding it, otherwise it will not take effect and the browser will slip into quirks mode.

Note that the DOCTYPE statement doesn't follow any of the syntax rules you just learned for writing HTML tags. Don't think of it as a part of the HTML as such, but as its own animal. Type it exactly as shown, with "DOCTYPE" in uppercase, adorned with an exclamation mark and unclosed, and you'll be fine.

There is no real need to use a doctype other than those already mentioned, but there are also doctypes for various versions of HTML, for an XHTML frameset, and also for XHTML 1.0 Transitional.

WHY NOT XHTML TRANSITIONAL?

XHTML Transitional is just that—a transition. It is designed to help developers make the move from one technical standard—HTML 4—to another technical standard—XHTML (Strict). This is a great learning step if you're stuck in your HTML 4 ways, but it shouldn't be seen as an ultimate goal. The difference between the Transitional XHTML and Strict XHTML is nothing more than the former allowing more tags and attributes than the latter. This might sound preferable, but in the long run it's not. XHTML Strict strips out most of the presentational crap that we're trying to get away from. By applying XHTML Strict we are helping to ensure that there is as little presentational junk in the markup as possible.

One increasingly unjustifiable reason why developers might opt for Transitional XHTML is if they have a need to accommodate older, rarely used browsers. Presentational elements might result in better presentation in browsers such as Netscape 4 but using such elements will be detrimental to the efficiency, and possibly accessibility, of your web pages.

Another reason might be if you are working with other, less knowledgeable people, or even completely handing over your code to someone (such as a client) who wants to add/alter/mangle it as they please. But in these cases, there's not much point in having a doctype at all (because, remember, quirks mode is for people who don't know what they're doing).

In fact, Transitional XHTML only makes sense when you don't have complete control over what you're doing. If you're not starting from scratch, or if you have to accommodate certain foibles or the whims of naïve project managers, for example, then you might not have much choice. And if you can use a doctype (and validate to it), it's better to use something than nothing at all.

But for the sake of argument, let's assume that we're not going to be handing over our Da Vinci to a manic toddler with a pack of crayons. Let's assume that we do have complete control over what we're doing (or at least striving to apply the highest standards). And let's assume that the best approach to web design is to completely separate structure and presentation (because, well, it is). And so let's assume that Strict XHTML is the way to go.

Language You should identify the primary language of a document either through an HTTP header ("HyperText Transfer Protocol"—it's a server thing—detail that is sent to the browser along with the HTML) or with the `xml:lang` attribute inside the opening `html` tag. Although this is not necessary to produce a valid HMTL document, it is an accessibility consideration. The value is an abbreviation, such as "en" (English), "fr" (French), "de" (German) or "mg" (Malagasy). Have a gander at *www.w3.org/International/articles/language-tags/* for more on the use of language codes.

The declaration for a document with primarily English content, for example, would look like this:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

After declaring a primary language, if you use languages other than that in your content you should further use the `xml:lang` attribute inline (such as `<span xml:lang="de">HTML Hund</span>`).

Content Type The media type and character set of an HTML document also need to be specified, and this is done with an HTTP header:

```
Content-Type: text/html; charset=UTF-8
```

The first part (in this example, the `text/html` bit) is the MIME type (Multipurpose Internet Mail Extension) of the file, and this lets the browser know what media type a file is and therefore what to do with it. All files have some kind of MIME type. A JPEG image is `image/jpeg`, a CSS file is `text/css`, and the type most commonly used for HTML is `text/html`.

The second part of the HTTP header (in this example, the `UTF-8`) is the character set.

Character sets include "ISO-8859-1" for many Western, Latin-based languages, "SHIFT_JIS" for Japanese, and "UTF-8," a version of Unicode Transformation Format, which provides a wide range of unique characters used in most languages. Basically, you should use a character set that you know will be recognized by your audience. So if the language is wholly English, for example, ISO-8859-1 is a code that is widely recognized. If there is a mix of languages, or a language that is not Latin based, the more general UTF-8 might be preferable. If the language is wholly Japanese and your target audience is also Japanese, SHIFT-JIS is the one to go for.

You can read more about character sets at *joelonsoftware.com/articles/Unicode.html*.

Perhaps the easiest way to set an HTTP header (or mimic it) is to use an "HTTP-equivalent" meta tag in the HTML, which would look something like this:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

All you need to do is pop that inside the `head` element (more on `head` and `meta` elements shortly), the browser will be able to work out the content type, and everything will come up smelling of roses.

HTML, Head, and Body

Right. So those are the all-important declarations out of the way. Now we can get on with applying those all-important HTML tags, using HTML to contain the two main page parts: `head` and `body`.

SETTING CONTENT TYPES SERVER-SIDE

The HTTP-equivalent `meta` tag does the job of setting a page's content type, but if at all possible it is preferable to use a genuine HTTP header. With the `meta` tag, the browser must receive the HTML file and then decipher the content type, but by establishing the content type on the server side before the HTML file is sent, the browser will be told what to expect beforehand.

One way of sending the content type is by using a server-side scripting language such as PHP:

```
<? header("Content-Type: text/html; charset= UTF-8"); ?>
```

If you don't want to (or can't) use a server-side scripting language, you might be able to go straight to the server with an ".htaccess" file. Most servers (Apache compatible) can have a small text file with the file name ".htaccess" that sits in the root directory and with the following line in it, you can associate all files with the extension ".html" with a MIME type and character set:

```
AddType text/html;charset=UTF-8 html
```

A basic page structure is going to look something like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
    <title>Uncle Jack's Sea Cow Farm</title>
</head>
<body>
<!-- A whole load of content -->
</body>
</html>
```

After the doctype declaration, we have `html`, which is the root element that specifies that the content of the document is HTML. It contains the remainder of the page information after the document-type declaration.

The first thing we find inside the `html` element is `head`. This is the header of an HTML document where information about the document is placed.

The `head` element comes straight after the opening `html` tag and contains information about the page that is not actually content. There are a few different elements that can go inside the `head` element, but one is required: `title` (we'll come to the `title` element, and those few others, in just a moment).

Finally, after the `head` element comes the all-important `body` element. This is the main body of an HTML document where all of the content is placed. This is the stuff that people will see, hear, or otherwise experience when they visit the web page.

HTML COMMENTS

You can place a comment anywhere in your XHTML like this:

```
<!-- Here's a comment -->
```

Absolutely nothing will change in terms of the visible or audible content—it is just a simple notice to anyone looking at the code of the web page.

Inside the Head... This isn't the place to go into much detail about the various tags that can be used inside the `body` element—there are many, and there's a lot to say about them, so you'll find more information about these in the rest of the book.

The `head` element is slightly different, however, with a much narrower scope of tags that can be used. `title`, `link`, `meta` and `base` are the four lonesome, specific tags described here. Other tags that can be used inside the `head` element are `style`, which is used to define page-specific CSS, and is explained in detail later in this chapter, and `script`, used to define page-specific scripts, such as JavaScript, which is given the coverage it deserves in Chapter 7.

`title` simply gives a title to the document. It will appear as the title of the browser window, as is also used for bookmarks.

```
<head>
    <title>Uncle Jack's Sea Cow Farm</title>
</head>
```

The `link` element defines a link to an external resource such as a CSS file, a shortcut icon, or customized navigation. There are a whole bunch of specific attributes that can be used (see the tag reference appendix for the details), but the most commonly used are `href`, which specifies the target of the link (much like text links, as will be explained in Chapter 2), and `rel`.

`rel` specifies the relationship of the target of the link to the current page. There are some universally understood values for the `rel` attribute, such as "shortcut icon" that browsers will recognize as the icon that should be used alongside the web address (in a "favorites" menu, for example) and "stylesheet," which browsers will recognize as the CSS file that should be linked to the page (see below). Some browsers will also allow the author to define customizable navigation elements with the link tag, such as next page, previous page, home, contact, etc., that will appear as options in the browser interface itself rather than the web page.

Here are a few examples of common uses of the `link` element:

```
<link rel="stylesheet" type="text/css" title="Some title"  href="/
somefile.css" />
<link rel="alternate stylesheet" type="text/css" title="Some
alternative title" href="/someotherfile.css" />
```

```
<link rel="shortcut icon" href="/favicon.ico" />
<link rel="next" title="Next page" href="nextpage.html" />
```

FIGURE 1.4 A "tool tip" will pop up when the cursor hovers over an element containing a title attribute.

The `meta` element specifies meta information, which is used to provide information about the HTML page (meta information being information about information).

We have already come across one type of `meta` tag—the "HTTP Equivalent" (see "Content type," above), but the simplest and most common form is a simply named meta tag, such as "keywords" or "author."

The important attributes that slot inside the `meta` tag are `content`, which is the meta information itself (and is therefore required), and `name`, which is, um, the name given to that information. `name` can be anything that tickles your fancy, but widely used examples are "keywords" and "description."

So, if you use meta tags, you might have a whole bunch of 'em, like this:

```
<meta name="keywords" content="fruit, banana, orange, apple,
kumquat, cucumber" />
<meta name="description" content="News, reviews and opinion on all
things fruity." />
<meta name="author" content="The Fruit Farmers Association of
Bujumburra" />
```

And then there's `base`. `base` defines the base location for links on a page. It isn't used that often, but in the interests of comprehensiveness, here's what it looks like:

```
<base href="/images/tootlepops/" />
```

If you were to slot this example into the head, what would happen is every file reference in the page would be in relation to "/images/tootlepops/". So, for example, `<img src="banana.jpg" alt="banana" />` would actually point to "/images/tootlepops/banana.jpg" and `<a href="morefruit/cucumber.html">Cucumber</a>` would point to "/images/tootlepops/morefruit/cucumber.html".

META TAGS: WHAT'S THE POINT?

The primary application of meta tags used to be in the optimization of a web page for search engine rankings. Keywords and descriptions were used by the search engine algorithms to judge how closely a web page matched a given phrase. Nowadays, however, few search engines take any notice of meta tags that specify page keywords due to their misuse and tendency to lead to irrelevant material. Instead, search engines tend to base their results on the page content itself. Google, for example, will only use the meta tag "description" to accompany a search result, but ignores meta tags altogether when it comes to judging a page's rank. It bases its results primarily on content, the page title, and also terms that are used to link to the page in question.

So is there any real point? Meta tags certainly aren't the force they used to be when it comes to search engines, but they can still be used to convey useful information about the page. Even if an application doesn't directly use a meta tag, someone looking at the page source itself could still benefit from such information. Another common value for the name attribute is "copyright," which won't be directly used by anything (such as search engines or browsers), but can be used to point out copyright information to a casual observer. On a large-scale site with multiple developers, you could also use meta tags to convey information about a page that means nothing to the outside world but can help internally.

The Generalist Tags—div and span

Throughout this book many tags will be mentioned, most of which have very specific purposes and are used to mark up very specific elements (such as images, tables, or quotes). There are two "generalists" that apply little meaning but are commonly used to group together sections of HTML and apply CSS to those groupings.

`div` is a division. It's a block-level element that groups together a chunk of HTML, and might look something like this:

```
<div id="content">
    <h1>How to make a falafel</h1> <p>Buy a falafel seed and plant
it in your garden.</p>
</div>
```

You will come across `div` more in Chapter 5, "Layout," where it is used to define navigation and content areas of a page, for example.

`span` is an inline element that groups together a chunk of inline HTML, such as single words or short phrases.

```
<h1>How to make a <span>falafel</span></h1>
```

`span` tags should be used sparingly because when a more meaningful tag can be used as an alternative (such as `em` for emphasis—see Chapter 2), that is more beneficial to the HTML structure. So, in the previous example, `<em>falafel</em>` would be better if you were actually attempting to emphasize the word *falafel*.

CSS Syntax

Although intrinsically linked with HTML in the formation of a web page, CSS, the language used for presentation of a page (see Introduction for more) has a completely different syntax, consisting of a collection of rules that are made up of *selectors*, *properties*, and *values*.

Rules

A typical CSS rule might look something like this:

```
h1 { font-size: 2em; }
```

Where:

"`h1`" is a selector, which defines which part of the HTML to apply the CSS to.

"`font-size`" is a property, which defines what specific presentational aspect of the targeted element you want to set.

"`2em`" is a value, which defines what the property should be set to.

"`font-size: 2em;`" is known as a statement.

And the whole lot, "`h1 { font-size: 2em; }`" is collectively known as a rule.

Selectors

Selectors specify which HTML elements the style declarations should be applied to. There are three main kinds of selectors: HTML selectors, id selectors, and class selectors.

HTML selectors simply specify an HTML element to which the declarations should be applied, so

```
h2 { color: red; }
```

Will make all `h2` HTML elements red.

id selectors attach styles to the HTML element with that corresponding id. So, if you had something like this in your HTML:

```
<h2 id="tree">Tree</h2>
```

And you just wanted to apply styles to that element, you would do this (note the # character at the start of the selector):

```
#tree { color: red; }
```

class selectors attach styles to HTML elements with a corresponding class. So, if you had something like this in your HTML:

```
<h2 class="plant">Tree</h2>
```

And you wanted to apply styles to that element and every other element with class=”plant”, you would do this (note the dot at the start of the selector):

```
.plant { color: red; }
```

You can attach a number of classes to an HTML element by separating them with spaces, such as:

```
<h2 class="plant leafy">Tree</h2>
```

Which will apply both of these rules:

```
.plant { color: red; }
.leafy { font-style: italic; }
```

WHAT SHOULD MAKE A CLASS OR ID NAME?

When choosing names for `id` and `class` attributes you should remember the structure and presentation separation. Once more, the values you choose should not suggest presentation.

Having:

```
<p class="redtext">Beach bottom bikinis...</p>
```

with the CSS to style it:

```
.redtext { color: red; }
```

is kind of missing the point. In this example, although `class="redtext"` doesn't actually do anything on its own, it is still not separating the suggestion of presentation from structure.

To put it in practical terms, what if you decided that you didn't want those particular classes in red any more? It would be a bit daft to then have:

```
.redtext { color: blue; }
```

So your class and id names should also be semantic, just like tag names.

```
<p class="bikinis">Beach bottom bikinis...</p>
```

This example makes much more sense to the structured semantic document and because it has nothing to do with presentation, you can apply any color or anything else to it and it will remain completely sensible.

And it's not just something as simple as colors or boldness, for example. Think whether the class or id names are suggesting what the corresponding CSS rule set is using and if they are they're not good.

`id="largebox"`, `class="hidden"`, `id="float1"` are all just as bad as `class="redtext"`.

Note that an id or class name cannot start with a number—it must begin with a letter or an underscore.

You can also be more specific as to which elements a class applies to by putting the class selector straight after another selector. If you only wanted to associate styles to an `h2` element with the class "plant," for example, then you could use the selector `h2.plant`. You could then use the selector `p.plant` to apply specific styles to `p` elements with the class "plant" and so on.

Pseudo-classes and Pseudo-elements

Pseudo-classes and pseudo-elements, which are bolted onto a selector with a colon, increase the specificity of a selector by adding a further condition, such as the first letter of an element (`whatever:first-letter`) or when the cursor moves over an element (`whatever:hover`).

The most widely used is probably `:hover`, which is applied like this:

```
a:hover { text-decoration: none; }
```

(This would cause a link's underline to disappear when it is hovered over by the cursor—see Chapter 3, "Links.")

There aren't many of these and most are very specific in what they do. You can find out more about what pseudo-classes and pseudo-elements do in Chapters 2, "Text," and Chapter 3.

Grouped Selectors

If there is a specific style you want to apply to more than one selector, there is no need to do something like this:

```
h2 { color: red; }
#kumquat { color red; }
.panda { color: red; }
```

To apply the same declaration block to more than one selector, all you need to do is separate selectors with commas, like this:

```
h2, #kumquat, .panda { color: red; }
```

Nested Selectors

You can directly target styles at nested HTML elements (elements within other elements) by specifying a space-separated list of parents before the desired element.

For example, if you wanted to apply a style to only those `em` elements that were within `p` elements, then you would do something like this:

```
p em { font-weight: bold; }
```

or if you only wanted to style `em` elements that were within `p` elements within an element with the `id` "content":

```
#content p em { font-weight: bold; }
```

and you don't have to specify every parent element. For example, if you wanted to style every `em` element in that "content" element:

```
#content em { font-weight: bold; }
```

Nested selectors often remove the need to apply id and especially classes as CSS hooks because you can target elements by their relationship to other elements. So, for example, if you wanted the links in a navigation area to be a different color than the links in the main content area, you could have something like this in your HTML:

```
<a href="this.html" class="prune">This</a>
<a href="that.html" class="prune">That</a>
<a href="theother.html" class="prune">The other</a>
<!-- etc -->
```

And then use this CSS:

```
.prune { color: orange }
```

But it would be much more sensible if you had something like this as the HTML:

```
<div id="navigation">
<a href="this.html">This</a>
<a href="that.html">That</a>
<a href="theother.html">The other</a>
<!-- etc -->
</div>
```

And then this as the CSS:

```
#navigation a { color: orange }
```

Which allows you to really cut down on HTML code.

SPECIFICITY

If you have two (or more) conflicting CSS rules that point to the same element, there are some basic rules that a browser follows to determine which one wins out. If the selectors are the same, then the one that is specified last in the style sheet will always take precedence. For example, if you had:

```
p { color: red; }

p { color: blue; }
```

`p` elements would be colored blue because that rule came last.

However, you won't usually have identical selectors with conflicting declarations on purpose (because there's not much point). Conflicts quite legitimately come up, however, when you have nested selectors. In the following example:

```
div p { color: red; }

p { color: blue; }
```

It might seem that `p` elements within a `div` element would be colored blue, seeing as a rule to color `p` elements blue comes last, but they would actually be colored red due to the *specificity* of the first selector. Basically, the more specific a selector, the more preference it will be given when it comes to conflicting styles.

The actual specificity of a selector takes some calculating, however, and isn't as straightforward as it might seem. There is a fixed way of calculating a selector's specificity and it goes like this:

You count the number of `id` attributes and call that number "a," then you count the number of other attributes (such as class selectors and pseudo-classes) and call that number "b," then you count the number of HTML selectors and call that number "c." Finally, you take a, b, and c, push them together and the number "a,b,c" is the overall specificity. Confused? A few examples might clear things up:

`p` has a specificity of 0,0,1 (a=0, b=0, c=1).

`div p` has a specificity of 0,0,2 (a=0, b=0, c=2).

`.tree` has a specificity of 0,1,0 (a=0, b=1, c=0).

`div p.tree` has a specificity of 0,1,2 (a=0, b=1, c=2).

`#baobab` has a specificity of 1,0,0 (a=1, b=0, c=0).

So if all of these examples were used, `div p.tree` (with a specificity of 0,1,2) would win out over `div p` (with a specificity of 0,0,2), and `#baobab` (with a specificity of 1,0,0) would win out over all of the others in this example.

It helps to keep the commas (having 1,0,0 rather saying "100") because it works on an infinite-base system (rather than base-10, which we commonly use). 11 class selectors ("0,11,0" rather than "0110"), for (an unlikely, in practice) example, is still less specific than one id selector ("1,0,0" rather than "100").

At-rules

At-rules use special types of selector that don't rear their heads all that often. CSS 2.1 has just three valid at-rules: `@import`, used to include one CSS file in another one (explained in more detail below); `@media`, used to assign a block of CSS to a specific media type such as screen or print (see Chapter 10, "Multiple Media"); and `@page`, which is used for paged media to apply properties to specific printed-page conditions.

Properties

So with the selectors you can associate styles to specific pieces of HTML. But what about the styles themselves? Properties are the presentational parts of an element that you can alter. There's a great deal of them, ranging from colors and font sizes to much more specific things such as `white-space` and `border-collapse`. The properties themselves will be covered in the relevant chapters.

EXTENDING SELECTORS: >, +, *, AND [X=Y]

The CSS standard allows a great deal of versatility and specificity with selectors. On top of grouping and nesting, in theory you can also have child selectors, universal selectors, adjacent sibling selectors, and attribute selectors. In practice, though, only universal selectors are widely supported (as in, supported by Internet Explorer 6).

Universal selectors (using the "*" symbol) match any and everything. So `form *` will target every box within a `form` element. An example of their use is in a user style sheet (see later in this chapter) to ensure that all text and backgrounds are a certain color by using something like `body * { color: black; background: white }`, which states that all boxes within the body should have black-on-white text.

Child selectors (using the ">" symbol) allow you to target the immediate descendant of an element, that is, the first nested element within a particular element. `body > p` will target `p` elements that are directly nested within the `body` element, but not `p` elements that are further nested in other elements, for example.

Adjacent sibling selectors (using the "+" symbol) allow you to apply a rule to an element that directly follows another element. `h1 + h2`, for example, will only apply an associated declaration block to `h2` elements that directly follow an `h1` element.

Attribute selectors will apply CSS to an HTML element with a specific attribute or attribute value. The syntax is *elementname[attributename=attributevalue]*. So, for example, `abbr[title]` will apply styles to `abbr` elements that have a `title` attribute (regardless of their value) and `abbr[title=Cascading Style Sheets]` will apply styles to `abbr` elements that have a title attribute with the value "Cascading Style Sheets." Instead of "=" you can also use "~=", which will match the selector to a word (of a space-separated list) within an attribute value (such as `abbr[title~=Cascading]`) or "|=", which will match the selector to the first "word" of a hyphen-separated list (designed primarily for the `xml:lang` attribute, where `abbr[xml:lang|=en]` will match both `xml:lang="en"` and `xml:lang="en-us"`, for example).

INHERITANCE

When you apply a property to a box, it will often be inherited by all the boxes contained within it as the descendants inherit styles from their ancestors. So, for example, when you apply a font size to the body element, blocks within the body (as in everything you see on the web page) will inherit that font size.

Not all properties are automatically inherited in this way. Dimensions, padding, borders, and margins, for example (see Chapter 5, "Layout"), will apply those elements to which they are explicitly applied. Although there is no general rule as to which properties are inherited by default and which are not, it is generally quite logical—it is likely that you will want colors and font sizes to be inherited more often than not, but highly unlikely that you would want all elements to be the same height as their parent element, for example.

Values

The values you can assign to different properties are often specific to each property (and will be covered throughout the chapters). But there are also values that are used by many different properties, namely units of measurement and color values.

Units

Units of measurement can be split into numbers, percentages, and lengths.

Unit		Suffix	Example
Number		[none]	`line-height: 1.5`
Percentage		%	`width: 80%`
Length	Em	em	`font-size: 2em`
	Pixel	px	`font-size: 16px`
	Point	pt	`font-size: 12pt`
	Pica	pc	`font-size: 10pc`
	Centimeter	cm	`width: 10cm`
	Millimeter	mm	`width: 100mm`
	Inch	in	`width: 2in`

ABSOLUTE AND RELATIVE UNITS

Absolute units are those that are irrevocably fixed no matter what the context—they are not dependent on anything. Meters and yards are examples of absolute units. In the case of CSS, `cm` (centimeters) and `in` (inches) are examples of absolute units.

Relative units, on the other hand, result in actual sizes that are *dependent* on (or relative to) something else. A percentage is relative, as is an em—the actual computed size of an object measured in such units depends on the situation in which they find themselves.

Whether pixels are absolute or relative is a contentious issue. Although they are popularly thought of as absolute units, they are relative, due to the fact that they can be different sizes depending on the size of monitor and screen resolution—a pixel could be 0.1 mm wide or it could be 10mm wide, for example. But in web design it is more helpful to think of pixels as absolute units, and that is how they will be treated throughout this book.

Sizes made up of relative units will have different computed sizes depending on browser preferences, such as text-size setting (ems) or the width of the browser window (%), but sizes made up of pixels remain fixed unless the user changes screen resolution, in which case all elements—text, images, layout, etc.—shrink or grow in real terms (as in their size in millimeters, for example) in the same relation to one another.

So because users do change things such as text size or window size and tend not to change their resolution (and even when they do, every relative size changes with it), the distinction between relative units (minus pixels) and absolute units (plus pixels) is much more helpful when it comes to understanding and manipulating the effects between the two different approaches.

"Absolute vs. Relative values" is discussed in Chapter 2 (in relation to text sizes) and in Chapter 5 (in relation to dimensions in layout).

An “em” represents the computed value of the font size. So if the text in a containing element is displayed at 16 pixels, then `1em` will be the equivalent of 16 pixels and `2em` will be the equivalent of 32 pixels, etc.

Note that the units come straight after the number, with no spaces, such as “12px” rather than “12 px.”

Color Values

Color values, which are used to color fonts, backgrounds, and borders, can be used to specify one of more than 16 million colors.

They can take the form of a hex (hexadecimal) value, an RGB (red, green, blue) value, or a color name.

Hex values use hexadecimal (once, and more accurately, known as “sexadecimal”) values, based on the base-16 number system (as opposed to the more familiar base-10 number system of decimal values), using digits from 0 to f (0 to 9 and then a to f).

Hex values are made up of a hash character (“#”) followed by either three or six hexadecimal characters. The three-digit version is essentially a compressed version of the six-digit version, where `#f00` is the same as `#ff0000` and `#c96` is the same as `#cc9966`, for example. The three-digit version is easier to decipher (the first character, like the first value in RGB, is red, the second green, and the third blue), but the six-digit version gives you finer control over the exact color. For example, this CSS rule specifies white text on a blue background:

```
p {
    color: #fff;
    background-color: #0000ff;
}
```

RGB values allow you to set decimal numeric values or percentages for the amount of red, green, and blue that make up a specific color. The three values within the RGB value can be from 0 to 255, 0 being the lowest level (for example, no red), 255 being the highest level (for example, full red). So, the previous example rule could also be written as:

```
p {
    color: rgb(255, 255, 255);
    background-color: rgb (0%, 0%, 100%);
}
```

There are 17 valid color names that can also be used. These are `aqua`, `black`, `blue`, `fuchsia`, `gray`, `green`, `lime`, `maroon`, `navy`, `olive`, `orange`, `purple`, `red`, `silver`, `teal`, `white` and `yellow`. You can also use the value `transparent`.

```
p {
    color: white;
    background-color: blue;
}
```

THE THREE STYLE SHEETS

There are three types of style sheet that get involved in the styling of a page: browser, author, and user.

The *browser style sheet* is that used by the browser to establish the default renderings of HTML elements. You should be able to rely on certain browser defaults such as `strong` elements being bold, or links being underlined, which, thankfully, means you don't have to specify every presentational aspect of every element on a page.

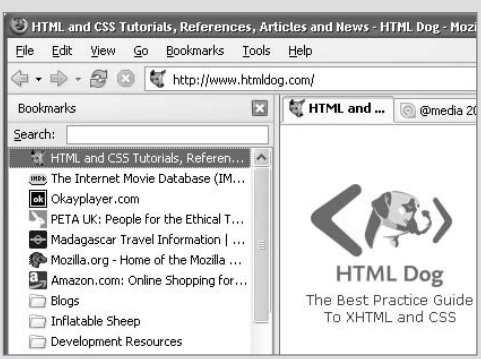


FIGURE 1.5 A page's title will appear in the top of the browser window, as well as in bookmarks. Also note the use of the link element to specify an icon, which appears next to the web address in the bookmarks and also in this browser's tabs.

The *author style sheet* is where you, as the page author, come in. This is the CSS that you apply to the HTML to make it look how you want. Rules in the author style sheet are given greater preference than the browser style sheet, but less preference than the user style sheet.

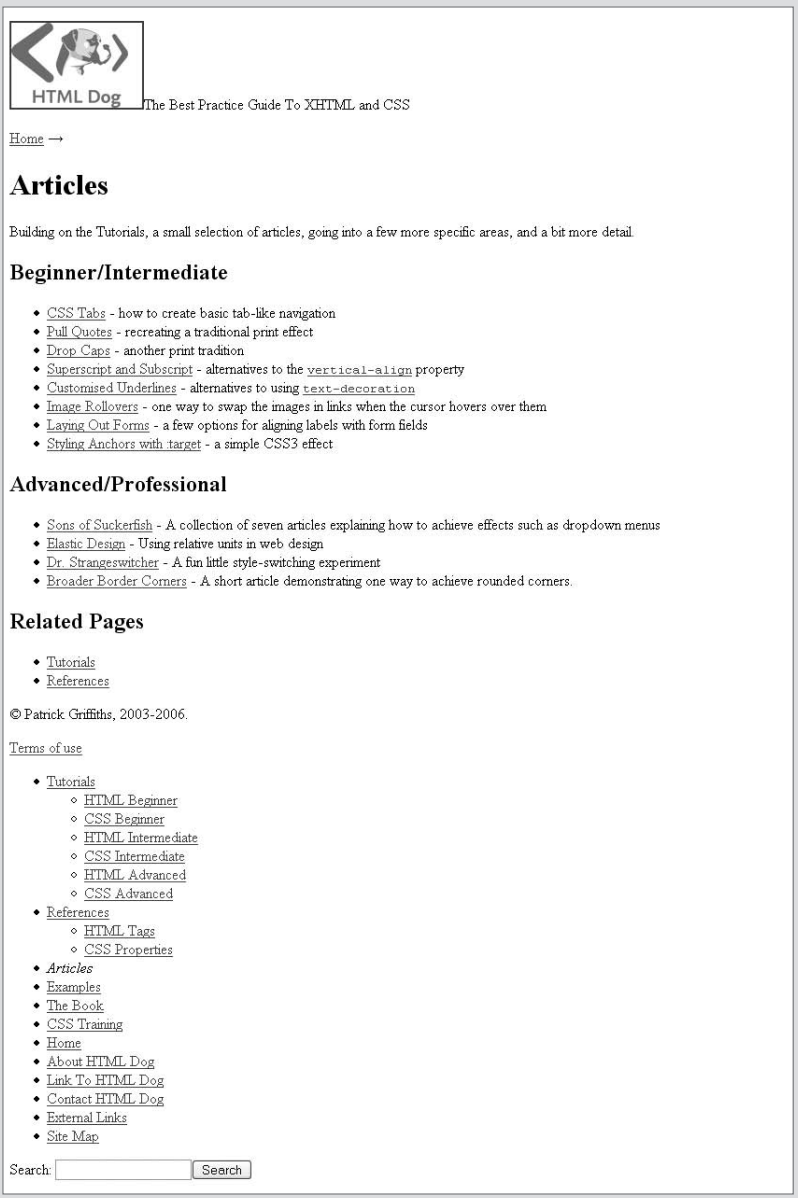

HTML Dog
The Best Practice Guide To XHTML and CSS
Home →
Articles
Building on the Tutorials, a small selection of articles, going into a few more specific areas, and a bit more detail.
Beginner/Intermediate
CSS Tabs - how to create basic tab-like navigation
Pull Quotes - recreating a traditional print effect
Drop Caps - another print tradition
Superscript and Subscript - alternatives to the vertical-align property
Customised Underlines - alternatives to using text-decoration
Image Rollovers - one way to swap the images in links when the cursor hovers over them
Laying Out Forms - a few options for aligning labels with form fields
Styling Anchors with :target - a simple CSS3 effect
Advanced/Professional
Sons of Suckerfish - A collection of seven articles explaining how to achieve effects such as dropdown menus
Elastic Design - Using relative units in web design
Dr. Strangeswitcher - A fun little style-switching experiment
Broader Border Corners - A short article demonstrating one way to achieve rounded corners.
Related Pages
Tutorials
References
© Patrick Griffiths, 2003-2006.
Terms of use
Tutorials
HTML Beginner
CSS Beginner
HTML Intermediate
CSS Intermediate
HTML Advanced
CSS Advanced
References
HTML Tags
CSS Properties
Articles
Examples
The Book
CSS Training
Home
About HTML Dog
Link To HTML Dog
Contact HTML Dog
External Links
Site Map
Search:
Search

FIGURE 1.6 A so-called unstyled page is actually applying the browser style sheet, which specifies things such as a serif font, underlining for links, and making heading elements large and bold.

The *user style sheet* is a style sheet that an individual user can apply to any or all web sites. Generally, this is for users with strong preferences or special needs. A user with poor eyesight might set a browser to always show content in a large type size, for example. Rules in the user style sheet are given greater preference than both the author and the browser style sheets.

Home
Search
HTML Dog
The Best Practice Guide To XHTML and CSS
Tutorials
HTML Beginner
CSS Beginner
HTML Intermediate
CSS Intermediate
HTML Advanced
CSS Advanced
References
HTML Tags
CSS Properties
Articles
Examples
The Book
CSS Training
Home
About HTML Dog
Link To HTML Dog
Contact HTML Dog
External Links
Site Map
Articles
Building on the Tutorials, a small selection of articles, going into a few more specific areas, and a bit more detail.
Beginner/Intermediate
CSS Tabs - how to create basic tab-like navigation
Pull Quotes - recreating a traditional print effect
Drop Caps - another print tradition
Superscript and Subscript - alternatives to the vertical-align property
Customised Underlines - alternatives to using text-decoration
Image Rollovers - one way to swap the images in links when the cursor hovers over them
Laying Out Forms - a few options for aligning labels with form fields
Styling Anchors with :target - a simple CSS3 effect
Advanced/Professional
Sons of Suckerfish - A collection of seven articles explaining how to achieve effects such as dropdown menus
Elastic Design - Using relative units in web design
Dr. Strangeswitcher - A fun little style-switching experiment
Broader Border Corners - A short article demonstrating one way to achieve rounded corners.
Related Pages
Tutorials
References
© Patrick Griffiths, 2003-2006.
Terms of use

FIGURE 1.7 With the author style sheet sitting on top of the browser style sheet.

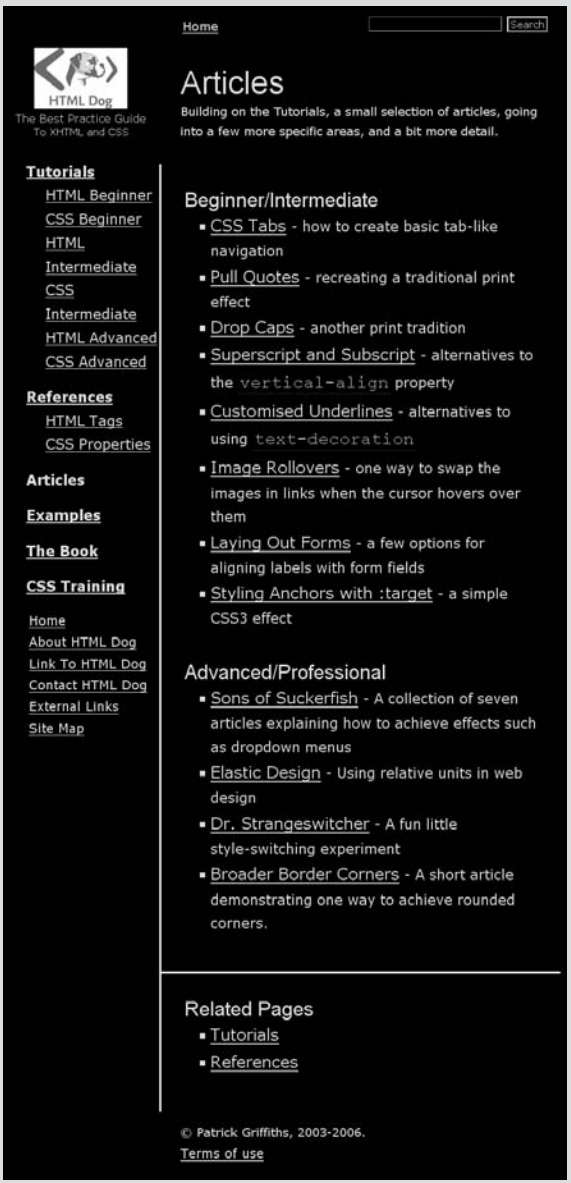

Home
Search
HTML Dog
The Best Practice Guide To XHTML and CSS
Articles
Building on the Tutorials, a small selection of articles, going into a few more specific areas, and a bit more detail.
Tutorials
HTML Beginner
CSS Beginner
HTML Intermediate
CSS Intermediate
HTML Advanced
CSS Advanced
References
HTML Tags
CSS Properties
Articles
Examples
The Book
CSS Training
Home
About HTML Dog
Link To HTML Dog
Contact HTML Dog
External Links
Site Map
Beginner/Intermediate
CSS Tabs - how to create basic tab-like navigation
Pull Quotes - recreating a traditional print effect
Drop Caps - another print tradition
Superscript and Subscript - alternatives to the vertical-align property
Customised Underlines - alternatives to using text-decoration
Image Rollovers - one way to swap the images in links when the cursor hovers over them
Laying Out Forms - a few options for aligning labels with form fields
Styling Anchors with :target - a simple CSS3 effect
Advanced/Professional
Sons of Suckerfish - A collection of seven articles explaining how to achieve effects such as dropdown menus
Elastic Design - Using relative units in web design
Dr. Strangeswitcher - A fun little style-switching experiment
Broader Border Corners - A short article demonstrating one way to achieve rounded corners.
Related Pages
Tutorials
References
© Patrick Griffiths, 2003-2006.
Terms of use

FIGURE 1.8 A user can also decide to lay his or her own style sheet on top, which could be used to aid accessibility by reversing foreground and background colors or making the text larger.

Applying CSS to HTML

So now you should have an idea of how CSS works, but it's pretty useless on its own. You need to attach it to the HTML—and there are a number of ways of doing this.

Inline CSS

Inline CSS is a "quick fix," sometimes used in testing but generally discouraged as a method of applying CSS.

It relies on the style attribute (which is actually deprecated, that is, outdated by something newer, and destined to be made obsolete, in XHTML 1.1) and goes a little something like this:

```
<p style="color: red;">Don't eat the pomegranate!</p>
```

Although this utilizes the language of CSS, it loses many of the large benefits that CSS is famous for—namely separating presentation from structure and the ability to make global style changes from a single source.

Embedded CSS

As mentioned earlier in this chapter, `style` is another tag that you can use within the head element, and it is used to apply page-specific, or embedded, CSS to an HTML page.

This method is much better than inline CSS (because it pulls presentation out of the body), but still doesn't completely separate structure and presentation (because there's still presentation in the HTML page). It should only be used when there are styles that will apply only to that single page.

The "bare bone" web page examples mentioned throughout this book largely use embedded CSS, mainly so you can see what is going on with greater ease but also because they are stand-alone single pages.

So, let's dip our toe in the HTML pool for a moment.

The `style` element is used to define CSS at a page level. This sits inside the `head` element, and its contents are simply a big ol' list of CSS rules, kinda like this:

```
<head>
    <title>Bujumburra</title>
    <style type="text/css">
    body {
        font-family: arial, Helvetica, sans-serif;
        color: black;
    }
    /* etc. etc. */
    </style>
</head>
```

Notice that inside the opening `style` tag is the `type` attribute, which tells the browser the content of the element. It's required, and the value of it, for our purposes, should be "text/css."

You can also use the `media` attribute, which states what media are associated with the styles. The value *can* be `aural`, `braille`, `embossed`, `handheld`, `print`, `projection`, `screen`, `tty` (teletype), or `tv` (television), but don't expect wide support for much more than `screen` and `print`. You could also have `media="all"`, but that's the same as not having any media attribute at all.

CSS COMMENTS

You can place a comment anywhere in your CSS like this:

```
/* Here's a comment */
```

Like an HTML comment, it won't add anything to the CSS rules, and is used just to provide a note for those looking at the code.

External CSS

External CSS is the kipper's knickers when it comes to applying CSS. It involves having the CSS rules in a completely separate file (with an extension of ".css"), to which HTML pages can link in a number of ways.

The first and most traditional way is by using the `link` tag, which needs to go inside the head element (see above) and would look like this (if the CSS file was called "somefile.css," for example):

```
<link rel="stylesheet" type="text/css" href="somefile.css" />
```

You can also manipulate an embedded style sheet to simply pull in an external style sheet using the `@import` at-rule. Instead of placing the CSS rules between the opening and closing `style` tags, you do something like this:

```
<style type="text/css">@import url("somefile.css");</style>
```

You should also be able to add a condition to the `@import` rule to target the style sheet at a specific media type, such as print:

```
<style type="text/css">@import url("somefile.css") print;</style>
```

Internet Explorer doesn't like this, but you can still have such a condition by applying a media attribute in the opening style tag, just like you can in a link tag:

```
<style type="text/css" media="print">@import url("somefile.css");
</style>
```

This `@import` method is sometimes used in preference over the `link` method because it hides the entire CSS file from browsers that don't have a firm grasp of CSS (such as Netscape 4.x) and could actually throw up an almighty mess. Such browsers will then simply apply their browser style sheet by itself, which, if the HTML is constructed properly, should be perfectly functional.

MULTIPLE STYLE SHEETS

Although on most occasions only one author style sheet is required and even preferable (holding all of a site's presentation in one file), it can be advantageous to have more than one. In the case of large, multisection sites, there may be a host of CSS rules that are only applicable to one section. If these rules are substantially large, it would be unnecessary for the rest of the sections to have to download them, or it could just be that it would be easier to manage this way (particularly if you have different developers for different sections). In such a case, you could have a CSS file specific to the section that is applied to the pages along with a core set of styles for the site.

There are a number of ways to apply multiple style sheets (such as simply having two `link` elements), but perhaps the easiest is to use the `@import` rule. An HTML page can simply link to the section style sheet and, within that style sheet, the core styles are brought in via `@import`. So, at the top of the section style sheet, you can simply have:

```
@import url("/css/core.css");
```

Just like in the method described for applying External CSS to a page (above), this will import another CSS file and essentially become a part of that one. There is no limit to the number of times this can be done (a style sheet can import another style sheet that imports two style sheets that each import another style sheet, for example) and they will all "cascade" into one another (just as "The Three Style Sheets" will) and live together as a happily intermarried family.

This page intentionally left blank

chapter 2

Text

IMAGES, MUSIC, ANIMATIONS, and even movies are splattered all over the Web in mind-boggling abundance. Amongst the squillions of bytes that make up the Web, though, the most common form of information is plain old text.

Without text you can't have hypertext, and without hypertext you can't have HyperText Markup Language. The *T* in HTML is fundamental to a web page, so it seems like a pretty good place to get the ball rolling and start putting something tangible on those pages.

In this chapter we'll first look at how to properly structure text, applying meaning with HTML, and then how that structured text can be manipulated to look exactly how you want it to look with CSS.

Structuring Text

Marking up text is an area where a lot of web designers have fallen into some bad practices. Before CSS came along, HTML had to be battered about to gain some rudimentary presentation, resulting in bloated, inaccessible, inflexible junk. Bad practices, which are still common, include the misuse of the `br` (break) tag to visually separate out chunks of text when `p` (paragraph) should be used, and bumping up the size and boldness of plain text so that it looks like a heading when a heading tag (such as `h1`) should be used. Keep in mind that although most browsers will render some elements in similar ways, it isn't what each tag makes the element look like that is important (or even significant), but rather what *meaning* they apply.

@media 2006
LONDON
15th-16th JUNE
Welcome Back!
The @media conference returns to London on 15th-16th June, bigger and better than before. It's the event of the year for anyone interested in learning about and discussing the latest approaches to web design with some of the world's most highly respected experts.
What?
@media, Europe's foremost professional web design conference, brings together some of the world's most highly respected web experts to talk about the latest major happenings, best-practice thinking, and cutting-edge techniques in the world of web design.
The presentations and panels will tackle a multitude of aspects of web design, covering topics such as user-interface design, web standards, CSS, DOM scripting, and accessibility.
The multi-track, two day conference schedule is further built upon to provide plenty of valuable networking time and evening activities.
Final Whistle
@media 2006 has come to a close, and can safely be declared a great success.
It will be back next year, but in the mean time, keep an eye on this web site for links to presentation slides, and podcasts.
Who?
Renowned speakers will be coming to @media from the USA, Canada, Sweden, Belgium, The Netherlands and the UK. Names you might recognise from their books, articles and general cutting-edge insight include Eric Meyer, Dan Cederholm, Andy Clarke, Molly Holzschlag, Jeffrey Veen, Jon Hicks, and Dave Shea to mention but a handful.
If last year is anything to go by, the speakers will be joined by delegates from all walks of web-life. Enthusiasts, freelancers, designers, developers, consultants, and project managers from charities, government departments, design studios, and multi-national corporations, not only from the UK, but from right across Europe, and beyond.
Where?
@media 2006 will be hosted at the prestigious QEII conference centre, in the heart of Westminster, in central London. It has great transport links with the UK, Europe, and the rest of the world and you will find plenty of hotels, bars, and restaurants in the area, not to mention attractions such as The Houses of Parliament and Westminster Abbey are almost literally on the venue's doorstep.
Register now!
"Book now to avoid disappointment!" Yes, we know everyone says that, but considering there are limited places, and last year's event sold out pretty quick-smart, we really mean it! Update: Too late – Sold out!
Banter
Oh, and by the way, we thought a little discussion corner would be a good idea so that you can let us know what you think about the conference plans, arrange meet-ups, or chat about what ever else you like.
Navigate
Home
Sessions
Venue
Last Year
News & Discussion
Delegate Area
Speakers
Schedule
Press & Buttons
Contact Us
Register
Last Year
@media 2005 was a great success. @media 2006 promises to be even bigger and better.
It was an incredible two days, and it felt much shorter than that. I'd wish it never stopped. High-level expertise blew my mind away.
Stéphane Deschamps
Sponsors
TITAN internet
.net
MAXDOX
@media 2006 is cheerily organised by Vivabit and is held together with the help of Event Wax
eventWax

FIGURE 2.1 The illustrations in this chapter are taken from the @media 2006 website (*www.vivabit.com/atmedia2006/*).

Although it may not seem necessary to mark up text elements such as quotations, abbreviations, and computer code (especially if no particular style is desired), by the same logic that the `p` tag should be used to mark up paragraphs, appropriate tags should also be used for other elements.

Basic Text Elements: Paragraphs, Line Breaks, and Emphasis

Perhaps the most important, fundamental text-related element is the paragraph (`p`), if only because it should usually make up the bulk of content in any text-rich web page.

```
<p>Greetings one and all. Welcome to the world of paragraphs.</p>
<p>This will be the second paragraph then...</p>
```

You've also got the `br` tag at your disposal, which can be used to insert a line break:

```
<p>Greetings one and all.<br />Welcome to the world of line
breaks.</p>
```

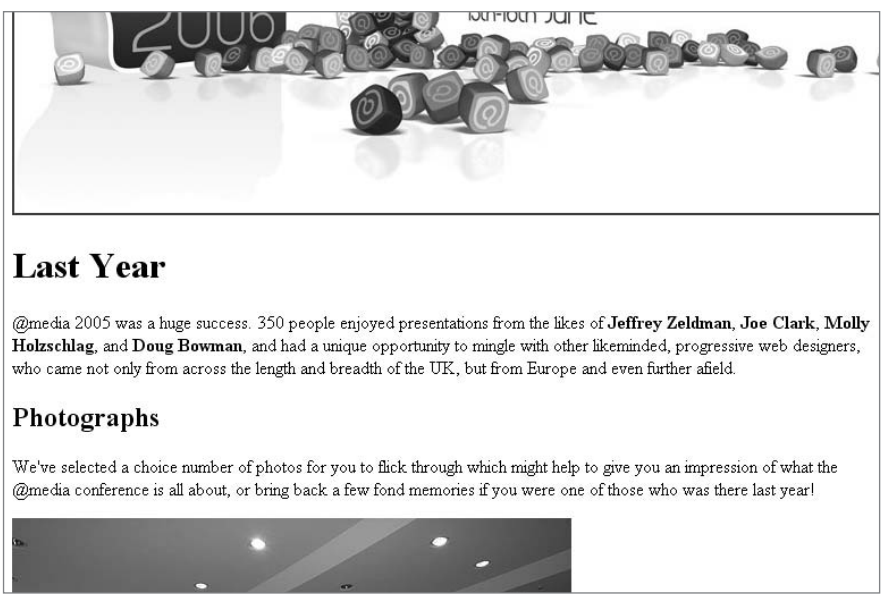

Last Year
@media 2005 was a huge success. 350 people enjoyed presentations from the likes of Jeffrey Zeldman, Joe Clark, Molly Holzschlag, and Doug Bowman, and had a unique opportunity to mingle with other likeminded, progressive web designers, who came not only from across the length and breadth of the UK, but from Europe and even further afield.
Photographs
We've selected a choice number of photos for you to flick through which might help to give you an impression of what the @media conference is all about, or bring back a few fond memories if you were one of those who was there last year!

FIGURE 2.2 A very common, simple page structure containing paragraphs and headings. Shown here naked, without the author style sheet, even the default browser style makes it clear straight away—we have chunks of text, which are paragraphs, and big bold headings. There's even a bit of strong emphasis going on in there.

TWO br S DO NOT A p MAKE

It is important to remember that a line break is not the same as starting a new paragraph, which has been a common misuse of the element—two line breaks may give the appearance of starting a new paragraph, but when we're talking about HTML, appearance doesn't mean a thing. A line break should only be used when there is a logical break in the flow of text, such as the new line of an address or a new line in a poem. As you will see later in this chapter, the look of paragraphs, including the spacing between them, can be manipulated, should be manipulated, and is easier to manipulate with CSS.

To emphasize text inside a paragraph—or other places, such as a heading (see next section) or in a table cell (see Chapter 8, "Tables")—there are two tags fit for the job: `em` (emphasis) and `strong` (strong emphasis):

```
<p>You lookin' at <em>me</em>? You lookin' at <strong>me</strong>?</p>
```

NOT i AND b!

`em` and `strong` are not replacements for the old presentational `i` (italic) and `b` (bold) tags. Although most browsers will display them in italics or bold by default, the important thing is that they apply meaning, whereas `i` and `b` apply only presentation (CSS's job). General emphasis is more than a visual thing—it can also be vocalized, for example, whereas bold and italics mean absolutely nothing to a blind person.

Headings

In most written documents, paragraphs make up the bulk of the content, but there's usually a need to break things up with headings and subheadings.

We've got six tags to play with here: `h1` (which is the highest level heading), `h2`, `h3`, `h4`, `h5`, and `h6` (the lowest level subheading).

www.htmldog.com/examples/headings1.html

The idea is to use the elements in order, with a single `h1` element for the page heading, then any number of `h2` elements for subheadings, any number of `h3` elements for sub-subheadings, and so on. They should be used in order, so, for example, an `h4` should be a subheading of an `h3`, which should be a subheading of an `h2`. And remember, don't worry if the default styling for a heading looks too big or too small—you can just use CSS to make it the size you want.

```
<h1>Headings</h1>
<p>This is all about headings.</p>
<h2>The First Subheading</h2>
<p>The first subheading was called Bob. Bob was a figurine cleaner
in a past life.</p>
<h2>The Second Subheading</h2>
<p>The second subheading was called Labella. She used to be a
chimney sweep.</p>
<h3>Labella's Chimney Sweeping</h3>
<p>Labella can still be persuaded to sweep chimneys for five beans a
chimney.</p>
<h2>The Third Subheading</h2>
<p>The third subheading was called John. He wasn't particularly
interesting.</p>
```

www.htmldog.com/examples/headings2.html

IF IT'S A HEADING, MARK IT UP AS SUCH

If something is a genuine heading then you should use one of these tags rather than styling a paragraph or other piece of text to simply look bigger, which has been a common bad practice in the past.

You will find that, by default, browsers will display these headings in bold with various sizes and spacing, but, as with paragraphs (and all other HTML elements), CSS can control all of these things so you can make them appear however you choose.

Quotations

As this part of the chapter progresses, you will find less commonly used tags that mark up very specific types of text as we go along. Just because they are used less frequently than paragraphs, emphasis, or headings doesn't mean that they're not important. Sticking with the ethos of applying meaning, you should always try to use these specific tags if you come across a piece of content that could be made more meaningful by using them. If you have a quotation or citation, for example, you should mark it up as such, using `blockquote`, `q`, or `cite`.

`blockquote` is used for a large, usually stand-alone, block-level quotation. Its content must be made up of other block-level elements, which in practice would usually be `p` elements.

If the source of the quote can be found online, you can supply a bit more information about the `blockquote` by using the `cite` attribute.

```
<blockquote cite="http://www.htmldog.com/guides/htmladvanced/text/">
<p>blockquote is designed to be for large, stand-alone quotations,
whereas q (quote) is used for smaller inline quotes.</p>
</blockquote>
```

QUOTATION SOURCES

As mentioned in Chapter 1, the Common `title` attribute can be used, and commonly is used, to show where a quotation or citation has come from when the `cite` attribute, the value of which should be a URI, is not appropriate.

`q` can also be used for smaller, inline quotes (and you can also use the `cite` attribute in the same way as used with `blockquote`).

In a mildly confusing way, there is also a `cite` tag, which can be used to mark up citations.

```
<p>So I asked <cite>Bob</cite> about quotations and he said <q>I
know as much about quotations as I do about pigeon fancying</q>.
Luckily, I found HTML Dog and it said...</p>
```

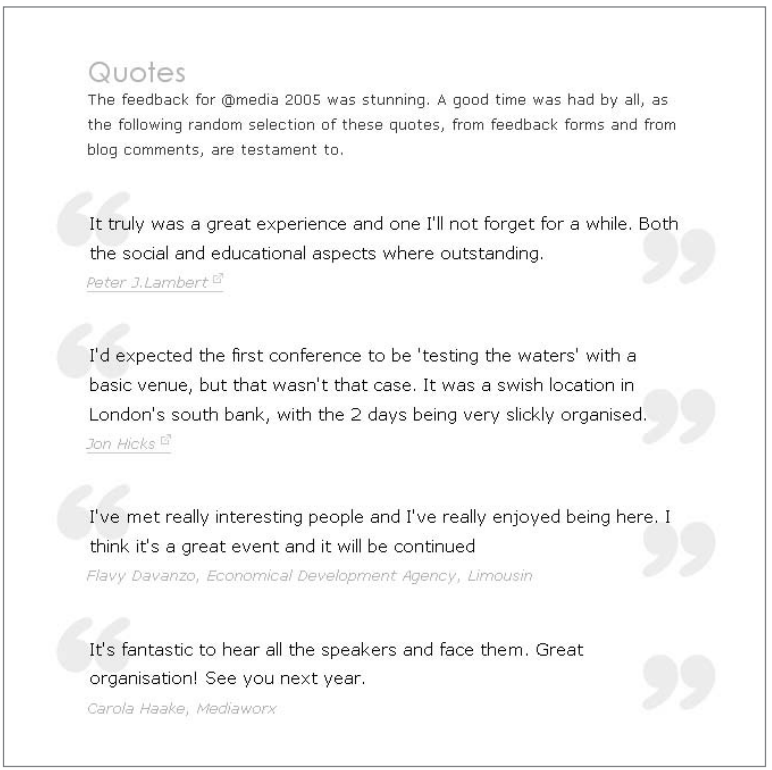

Quotes
The feedback for @media 2005 was stunning. A good time was had by all, as the following random selection of these quotes, from feedback forms and from blog comments, are testament to.
It truly was a great experience and one I'll not forget for a while. Both the social and educational aspects where outstanding.
Peter J.Lambert
I'd expected the first conference to be 'testing the waters' with a basic venue, but that wasn't that case. It was a swish location in London's south bank, with the 2 days being very slickly organised.
Jon Hicks
I've met really interesting people and I've really enjoyed being here. I think it's a great event and it will be continued
Flavy Davanzo, Economical Development Agency, Limousin
It's fantastic to hear all the speakers and face them. Great organisation! See you next year.
Carola Haake, Mediaworx

FIGURE 2.3 When it's a quotation, it's gotsta be marked up as a quotation. Underneath these pretty little representations of quotes are blockquotes, with the name of the person who made the quote marked up in a cite element.

Abbreviations and Acronyms

Again, we're getting specific, but if you come across abbreviations, you should mark them up as such.

abbr can be used for abbreviations—a shortened form of a word or phrase. *HTML* is an abbreviation, as is *CSS*, for example.

You can also use **acronym**, but keep in mind that an acronym is much more specific—it is a *pronounceable* abbreviation that is made up of the initial

letters or parts of words of that phrase. *NATO* is an example of an acronym, as is *UNICEF*.

```
<p>Scientists at <acronym title="National Aeronautics and Space
Administration">NASA</acronym> were attempting to teach Jiminy
the locust <abbr title="HyperText Markup Language">HTML</abbr>.
They seemed to overlook the fact that he was a <abbr title="Dumb
insect who couldn't comprehend what a computer was, let alone use
one">DIWCCWACWLAUO</abbr>, however.
```

NOT ALL ABBREVIATIONS ARE ACRONYMS!

An acronym is a form of abbreviation, but an abbreviation is not necessarily an acronym. *HTML* and *CSS* are not acronyms because they are not (supposed to be) pronounceable.

FIGURE 2.4 All hail the tool tip! If an abbreviation has a title attribute, its value can be used to state the full phrase that the abbreviation represents.

Preformatted Text and Computer Code

If you want to mark up code in your HTML, there's a tag just for the job, and it's `code`.

```
<code>nascaristhedullestofallmotorsports=true;</code>
```

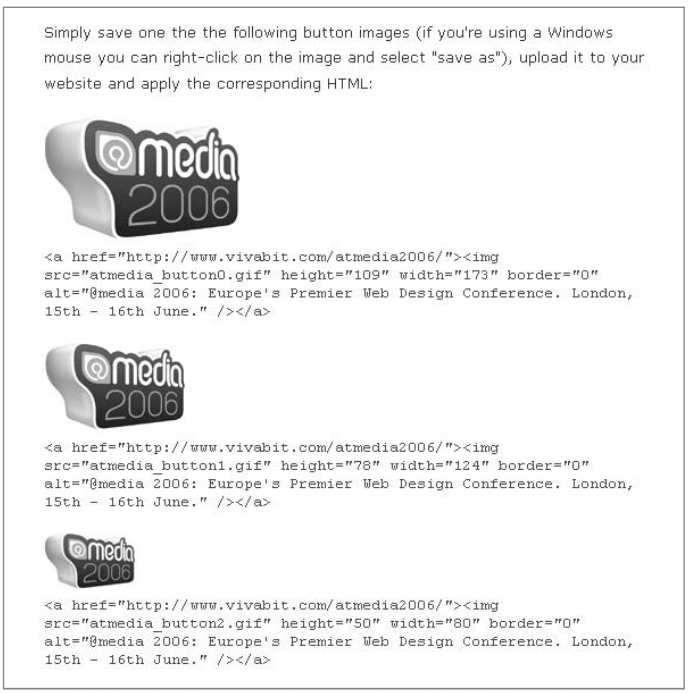
There's even a tag, `var`, for variables in computer code:

```
<code><var>nascaristhedullestofallmotorsports</var>=true;</code>
```

`samp` is another close relation of `code`, and defines sample output, from a computer program, for example:

```
<p>The result will either be <samp>Kid</samp> or <samp>Koala
</samp>.</p>
```

Most elements don't take too much notice of white space, that is, things such as spaces, tabs, and carriage returns. In a `p` element, for example, if there are places in the text where there are consecutive spaces, or if you start new lines, the end result will actually be truncated—the browser doesn't see much meaning in white space.


Simply save one the the following button images (if you're using a Windows mouse you can right-click on the image and select "save as"), upload it to your website and apply the corresponding HTML:

```
<a href="http://www.vivabit.com/atmedia2006/"><img
src="atmedia_button0.gif" height="109" width="173" border="0"
alt="@media 2006: Europe's Premier Web Design Conference. London,
15th - 16th June." /></a>
```

```
<a href="http://www.vivabit.com/atmedia2006/"><img
src="atmedia_button1.gif" height="78" width="124" border="0"
alt="@media 2006: Europe's Premier Web Design Conference. London,
15th - 16th June." /></a>
```

```
<a href="http://www.vivabit.com/atmedia2006/"><img
src="atmedia_button2.gif" height="50" width="80" border="0"
alt="@media 2006: Europe's Premier Web Design Conference. London,
15th - 16th June." /></a>
```

FIGURE 2.5 If it's computer code, ladies and gentlemen, it belongs in a code element. Simple.

The `pre` element is slightly different—spaces, tabs, and line breaks become as important a part of the content as the letters, numbers, and other characters, and the default browser rendering of displaying this white space (unlike the default rendering of most elements) reflects this.

`pre` is most commonly used to mark up blocks of computer code (where indentations, etc., can be meaningful and important).

```
<pre><code>
&lt;div id="intro"& gt;
```

```
    &lt;h1&gt;Some heading&lt;/h1&gt;
    &lt;p&gt;Some paragraph paragraph thing thing thingy.&lt;/
p&gt;&lt;/div&gt;
</code></pre>
One last oddity is kbd, which is used to specifically suggest text
that should be entered by the user:
<p>Now type <kbd>kumquat</kbd>.</p>
```

Editorial Insertions and Deletions

Have you ever used "Track Changes" in Microsoft Word? It can be used to show insertions and deletions, which can be helpful when more than one author is working on the same document. HTML documents are just that: documents, much like Word documents. Although probably less useful than in word processing, HTML has equivalents of "Track Changes" and they are `ins` (for insertions) and `del` (for, wait for it… deletions).

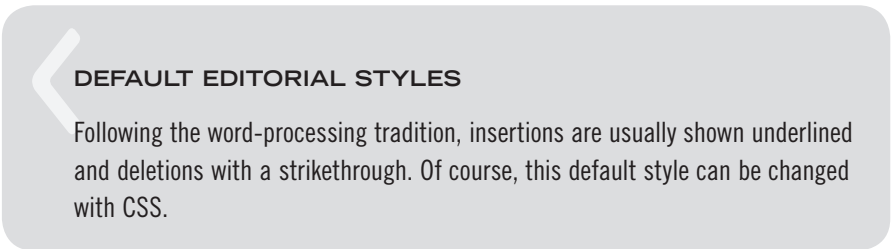
You can use the `datetime` attribute, the value of which (in the format of YYYYMMDD) would explain when the insertion or deletion was made. Like quotes, you can also use the `cite` attribute, which, in this case, is intended to point to information explaining why the given change was made.

```
<p>Patrick was walking down the road when he saw a <del datetime="2
0040329">fluffy kitten</del><ins cite="http://www.htmldog.com">giant
rabid snarling mutant saber-toothed goat</ins>.</p>
```

It's worth noting that `ins` and `del` are peculiar in that they can be used as either inline (such as in the previous example) or block elements (containing multiple paragraphs, for example). However, you should note that when `ins` and `del` are being used as inline elements, they cannot contain block-level elements. For example, the following would not be legal:

```
<p>
<ins><div>giant rabid snarling mutant saber-toothed goat<div></ins>.
</p>
```

Remember (from Chapter 1, "Getting Started") that you still can't put block elements inside inline elements, so if you're intending an `ins` element to be inline, for example, then it can't contain block elements.


DEFAULT EDITORIAL STYLES
Following the word-processing tradition, insertions are usually shown underlined and deletions with a strikethrough. Of course, this default style can be changed with CSS.

Multilanguage and Bidirectional Text

As explained in Chapter 1, the `xml:lang` attribute can be used in just about any HTML tag to define the language of its content, but sometimes a language will also need to be read in a different direction from its surrounding content.

The `bdo` tag can also be used to define bidirectional text, such as languages that are read in a different direction from the default language (Hebrew in an English document, for example).

The `dir` attribute, which is required, is used to define the direction of the text, and its values can be `ltr` (left-to-right) or `rtl` (right-to-left).

```
<bdo dir="rtl">smug desserts</bdo>
```

Addresses

…and there's even a tag to mark up your address.

`address` is very specifically intended to mark up the contact details, such as a street address, for a page, or major part of a page (such as a contact form).

```
<address>
HTML Dog House<br />
HTML Street<br />
Dogsville<br />
The Oligarchic Republic of Dogland
</address>
```

Styling Text

Excellent. So now you're a master in the art of structuring text with HTML. If you view these elements in a browser as they are, with their default visual rendering, it's quite probable that they're not exactly how you would want them to look. The browser can only go so far in establishing some basic styles; the rest is up to you and your friend CSS. With the text styling properties outlined here you can take full control of font types, sizes, colors, and spacing.

Fonts

Because a web page with lots of different fonts looks about as hot as a hippopotamus with a skin complaint, you probably won't find yourself wanting to change the font style—using `font-family`—that often. In most circumstances, you would apply it to the body, setting the base font for the entire page, and then maybe sparingly on some specific elements.

The value of `font-family` can be a single font name (which, if it is made up of more than one word, should be written in quotation marks), or multiple font names, separated by commas. By doing this, if a browser cannot find the first choice font, it will move on to the next in the list. This is used to provide a backup for when preferred fonts fail or for when similar fonts with different names appear on different operating systems. For example, traditionally you would find the `Arial` font on PCs but `Helvetica` on Macs, so specifying these fonts would cover both adequately. Finally, it is a good idea to back up the pack with a generic font keyword such as `serif`, `sans-serif` or `monospace` in case all else fails.

```
body { font-family: "Times New Roman" }
h2 { font-family: arial, helvetica, sans-serif }
```

A BROWSER CAN ONLY DISPLAY FONTS INSTALLED ON THE USER'S COMPUTER

You might have 3.2 billion fonts gathered from a stack of magazine cover discs and downloaded from the Internet, but if those looking at your web pages don't have the same fonts installed on their computer then the computer simply won't be able to apply them. You need to be careful which fonts you specify. There are certain "safe" fonts that most users will have on their computers—they will probably have Arial or Helvetica but you're probably pushing it if you count on Slug Invader Hieroglyphics or Curly Gothic Roman Dings Bold Condensed 5.

If you really want to use a relatively obscure font, you can use the comma-separated value to specify the very specific font on the off chance that a user will have it but providing a backup by specifying safer fonts for the browser to fall back on. It is probably a good idea to test the web pages in the backup fonts in situations like this, though...

```
body { font-family: "Slug Invader Hieroglyphics", arial, helvetica,
sans-serif }
```

Welcome Back
The @media conference returns to London on 15th
and better than before. It's the event of the year fo
interested in learning about and discussing the lates
web design with some of the world's most highly re
What?
@media, Europe's
foremost professional
Final W

FIGURE 2.6 There are a number of fallbacks going on here to achieve a semi-common font. Most Windows PCs are armed with the Century Gothic font, and most Macs have the similar Avant Garde installed. So with something along the lines of `h1, h2 { font-family: "Century Gothic", "Avant Garde" }`, the headings get a little bit of special treatment compared to the rest of the text (which is set by `body { font-family: Verdana, Geneva, Arial, Helvetica, sans-serif }`).

Color

As already mentioned in Chapter 1, the `color` and `background-color` properties can be used to specify the colors of just about anything on a web page. Although `background-color` can be used on other elements as well as textual elements, the `color` property, which means the foreground color, essentially applies only to text (although it can also apply to borders—see Chapter 5, "Layout").

```
body {
    font-family: "Times New Roman", Times, serif;
    color: white;
    background-color: black;
}
code { color: #900; }
blockquote { background-color: #efe; }
```

BASE COLORS

Like `font-family`, it is common practice to apply a color to the body, which will set the base text color of the page, being inherited by text throughout. Individual elements can then, of course, be colored separately if needed.

Size

`font-size` sets the font size. Well, duh.

The most commonly used units for computer displays are `px`, `em`, `%`, and keywords (such as `small`, `medium`, or `large`—see the CSS Property Appendix for the whole lot), although things are done slightly differently when it comes to other types of media (see "Styling for Print" in Chapter 10, "Multiple Media").

```
body { font-size: 80%; }
h1 { font-size: 2em; }
```

SIZE EXAMPLE

Figure 2.6 is also a good example of font sizes. The page kicks off with a font size of 80 percent. `h1`s are then set to 4ems, `h2`s to 2ems, and `h3`s to 1.5ems

ABSOLUTE VS. RELATIVE VALUES I

So which unit should you use to size text on a computer screen—an absolute unit such as pixels, or a relative unit such as ems?

Ems. It's as simple as that. Pixels should not be used to size text.

Well, OK, it's not as simple as that. Actually, any unit relative to the screen size (so that's not pixels, no matter who tries to tell you a pixel is a relative unit) is OK—percentages and font-size keywords will do just as nicely.

The reason? Because these units allow text to be elastic—to expand or contract depending on the user's text-size settings. It accommodates a user's preferences.

When CSS came along and opened up the control over a web page's visuals, many grabbed on the fact that for the first time they could use pixels to define a font's size. Such pixel-perfect control sounds like a good thing—as a designer you have complete control over what the final thing will look like. It is more likely that what you see on your computer screen is what users will see on theirs. But that's kind of missing the point of the Web, which is such a flexible beast.

Here's the crux of the matter: Text sized using pixels is less accessible than text sized using ems. Usability should be enough to convince—if users prefer something a certain way (such as larger or even smaller text) then they are going to respond better if their preference is accommodated. But it's not just about usability and preference. To many it is about necessity: If a user cannot comfortably read text, then he or she will benefit by being able to resize that text. Accessibility isn't only about blindness and screen readers—visual impairment

is enough to warrant such attention. And visual impairment is something that we will all encounter as we grow older.

A web page on a computer screen is not a printed piece of paper. Dimensions can be changed and user preferences can be accommodated. It's a great feature. Most acknowledge this, but even those who did continued to use pixels because they felt that there was no alternative.

The problem lay in Internet Explorer. The first problem (for pixel-fan designers, anyway) was that whereas browsers such as Mozilla offered a "text-zoom" function, which increased the size of even pixel-sized text, IE did not. To IE a pixel is a pixel is a pixel. If the designer specifies pixels, the browser delivers pixels. So when the user opts to change the text size setting, nothing happens to the pixel-sized text.

So to accommodate Internet Explorer's text-sizing accessibility feature, relative units are the only option. But here comes the second problem...

When you specify fonts in ems, for example, the jumps between sizes in Internet Explorer are so large that on the "smaller" setting text becomes unreadable. And because "smaller" is a popular setting, you are hindering accessibility—even to people with 20/20 vision, let alone anyone else.

But then percentages came to the rescue. By setting the initial font size of the body in a percentage and then in ems thereafter, the jumps between IE's text-size settings become smaller, and everyone, from those who browse at "smallest" to those who browse at "largest," can be happy.

```
body { font-size: 80%; }
h1 { font-size: 2em; }
h2 { font-size: 1.5em; }
```

At the end of the day there are real benefits to sizing text using relative units and there are no real reasons not to.

FIGURE 2.7 Because the font sizes are all set in ems (after an initial setting in percentages), users can bump up the size of the text, if they so choose, which is a great accessibility benefit. If they were set in pixels, the majority of web surfers, using IE, would not have this benefit.

You can read more about "Elastic Text" at *www.htmldog.com/articles/elasticdesign/*, which also covers the pros and cons of using absolute and relative values in layout—a technique that will be looked at in Chapter 5.

Line Height

You can adjust the height of the lines in text, such as a paragraph, without adjusting the size of the font, just like line spacing in a word processor. `line-height` is handy little critter—if used wisely, it can make your text much more readable.

```
p { line-height: 1.5 }
```

SUFFIXLESS

Line-height is one of only two properties (`z-index` being the other—see Chapter 5) that do not require a suffix (although you can specify any length or percentage, if you want). `line-height: 1.5` is the same as `line-height: 1.5em`.

into design practices in trends such as Ajax, syndication, tagging,
progressive enhancement, and anticipatory interaction, philosophical
shifts in business towards openness, generosity, and collaboration with
users.
Designing the Next Generation of Web Apps will be presented by Jeffrey Veen
The Fine Art of Web Design
Andy Clarke is a designer who sees web pages wherever he goes. From paintings on the walls of Tate Modern to posters on the walls of tube stations. He sees web pages everywhere, leaping out from the pages of magazines and newspapers, record covers, political posters. It might be that Andy just needs to remember to take his medication, but he finds inspiration for innovative designs in the most unlikely of places.
In his presentation, Andy will take you for a ride around his world, showing how he gains inspiration for layouts and designs which break with the boxey conventions which are sometimes associated with web standards and how we can use advanced CSS to implement them in progressive and accessible ways.
The Fine Art of Web Design will be presented by Andy Clarke
Fine Typography On the Web
With a technological revolution and 500 years of typographic tradition under our belts, why in the world has the Web been stuck with Verdana and Arial for so long? There are now multiple ways to include custom fonts in an HTML document, and each has its advantages and

FIGURE 2.8 The base line-height of the @media 2006 site is set at 1.7, as shown in the rest of the figures in this chapter. If left at the default, the text is a bit more squashed and a bit more difficult to read.

Bold and Italics

Don't forget that we're not using HTML for presentation anymore—not even `b` or `i` tags are allowed. When we want to set the thickness or slant of text, there's CSS that does the job much better.

`font-weight` sets the boldness of a font. By far the most commonly used and practical values are `bold` or `normal`.

```
a { font-weight: bold }
h2 { font-weight: normal }
```

Text can be italicized (or de-italicized) using `font-style`, using `italic` (or `normal`) as the value.

```
h1, h2 { font-style: italic }
```

Upper and Lower Case

If you really want to get hard core (and why not?) you shouldn't even be writing words all in capitals in your HTML just to achieve a certain look for a heading, or emphasis, for example. Instead of `<h1>THIS IS A DAMNED FINE HEADING</h1>`, why not try `<h1>This is a damned fine heading</h1>` and trying out `font-variant` or `text-transform`?

`font-variant: small-caps` will convert lowercase letters to small uppercase letters, but that's probably not as useful as `text-transform`, which properly (no pussy-footing around with "small caps" here) converts the case of letters. Values for `text-transform` can be `capitalize` (which capitalizes the first letter of every word), `uppercase` (every letter uppercase), or `lowercase` (every letter lowercase).

```
h1, h2 { text-transform: uppercase }
```

www.htmldog.com/examples/case.html

The `font` Shorthand Property

We're going to come across "shorthand" properties a few times throughout this book. They're great little shortcuts that enable us to define a number of styles (which can otherwise be defined separately) in one property, so reducing code. They might be more confusing to read than the separate, specific properties at first, but the more you get used to them, the snappier your code will be.

`font` can be used to specify italics, small-caps, boldness, size, line-height, and font name all in the one property. The value is, essentially, a combination of `font-style`,

font-variant, font-weight, font-size, line-height, and font-family, and is used in the format of *font: font-style font-variant font-weight font-size/line-height font-family*. Only the font-size and font-family parts are required.

```
p { font: italic small-caps bold 0.8em/1.5 arial, Helvetica,
sans-serif }
.booba { font: bold 3.5em arial, helvetica, sans-serif }
```

font WILL RESET PREVIOUS STYLES

Before applying any values, font will reset any previous font-style, font-variant, font-weight, font-size, line-height, or font-family values to their initial settings. So for example...

```
.wooha {
  font-weight: bold;
  font-variant: small-caps;
  font: 2em arial;
}
```

...will not apply bold or small-caps because font comes along after the font-weight and font-variant declarations and unashamedly resets them.

Underline and Strikethrough

text-decoration can be used to underline, overline, strikethrough, or turn off any existing decoration (such as on links) using the values underline, overline, line-through, or none.

```
ins { text-decoration: none }
```

UNDERLINE WITH CAUTION

Underlined elements are commonly recognized as links (or insertions, if text is in an editorial context), so you need to be careful about where you apply this style. The practice of styling acronyms and abbreviations with dotted underlines (using border-bottom: 1px dotted) has become a popular way of discerning them from the crowd, but there is a possibility that even this will be confusing. See Chapter 3, "Links," for more, including how to make your own custom underlines with borders and images.

DON'T BLINK!

Even if it did work in IE, it would be an extremely bad idea to use `text-decoration: blink`. Blinking text is notoriously disliked by users and is BAD from an accessibility point of view—and not in a Michael Jackson way, either.

Letter and Word Spacing

OK… now there's a danger of getting carried away with all of this power CSS is giving us. If used willy-nilly, things such as the self-descriptive `letter-spacing` and `word-spacing` can make your page look a mess, but, as always, use it wisely (maybe for something a little different than the norm in headings, for example), and the results could be rewarding.

```
p {
    letter-spacing: 0.3em;
    word-spacing: 1em;
}
```

Indenting

Another property rooted in traditional print styling that doesn't translate quite as well on a web page (because it's not really a convention on the Web) is `text-indent`, which indents the first line of text in a box by a length or percentage.

```
p { text-indent: 1em }
```

www.htmldog.com/examples/textalign.html

Horizontal Alignment

`text-align` horizontally aligns text within a block box, such as a default paragraph, to the `left` (which is usually the default), `right`, or `center`; or `justify`.

```
p { text-align: right; }
```

Another word of warning: Left-aligned text is easier to read on the Web than justified text, so use the justify setting sparingly.

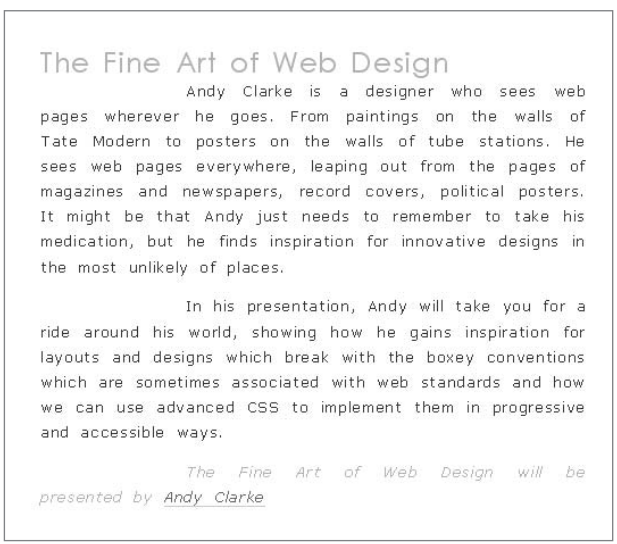

The Fine Art of Web Design
Andy Clarke is a designer who sees web pages wherever he goes. From paintings on the walls of Tate Modern to posters on the walls of tube stations. He sees web pages everywhere, leaping out from the pages of magazines and newspapers, record covers, political posters. It might be that Andy just needs to remember to take his medication, but he finds inspiration for innovative designs in the most unlikely of places.
In his presentation, Andy will take you for a ride around his world, showing how he gains inspiration for layouts and designs which break with the boxey conventions which are sometimes associated with web standards and how we can use advanced CSS to implement them in progressive and accessible ways.
The Fine Art of Web Design will be presented by Andy Clarke

FIGURE 2.9 Going crazy with letter-spacing, word-spacing, text-indent, and text-align.

Vertical Alignment

`vertical-align` is not quite as exciting as it might at first sound (because I just know you were getting excited there). It applies only to inline boxes (usually text), and is not meant to align chunks of a page vertically. In many cases, as explained later, there are better alternatives to using `vertical-align`.

Values `top`, `middle`, `bottom`, `text-top`, `text-bottom`, `sub` (subscript), and `super` (superscript) rely on the styled box being smaller than some or all of the text in the rest of the line (otherwise it will already be at all of those positions).

A length or percentage can also be used.

```
.power {
    font-size: 80%;
    vertical-align: super;
}
```

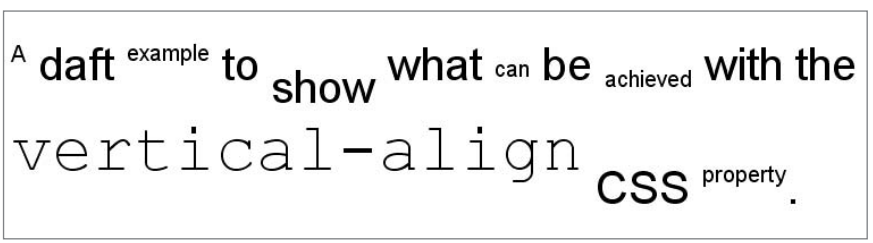


FIGURE 2.10 Using the vertical-align property can push parts of the text to any degree up or down, but doing this will alter the height of the line such an effect sits on. See *www.htmldog.com/examples/verticalalign.html*.

You can also achieve vertical alignment by using positioning (see Chapter 5 for more), which is a tad more complicated, but gives you a smidgen more control. Take a glance at the *Superscript and Subscript* article (*www.htmldog.com/articles/superscript/*) and corresponding example (*www.htmldog.com/examples/superscript.html*) on the HTML Dog website for more.

More Text Styling Techniques

To achieve more specific traditionally print-related styles, you can expand your text-styling options by playing around with other CSS properties (particularly those covered in Chapter 5).

You might have a penchant for drop caps, for example:

- www.htmldog.com/articles/dropcaps/
- www.htmldog.com/examples/dropcops1.html, dropcaps2.html, and dropcaps3.html

Or you might fall giddy with glee at the mere waft of a pull-quote or two:

- www.htmldog.com/articles/pullquotes/
- www.htmldog.com/examples/pullquotes1.html, pullquotes2.html, and pullquotes3.html

chapter 3

Links

WE'VE JUST COVERED the *T* in HTML, but it's links that give it the *HT*—HyperText is the method of moving between places by selecting links.

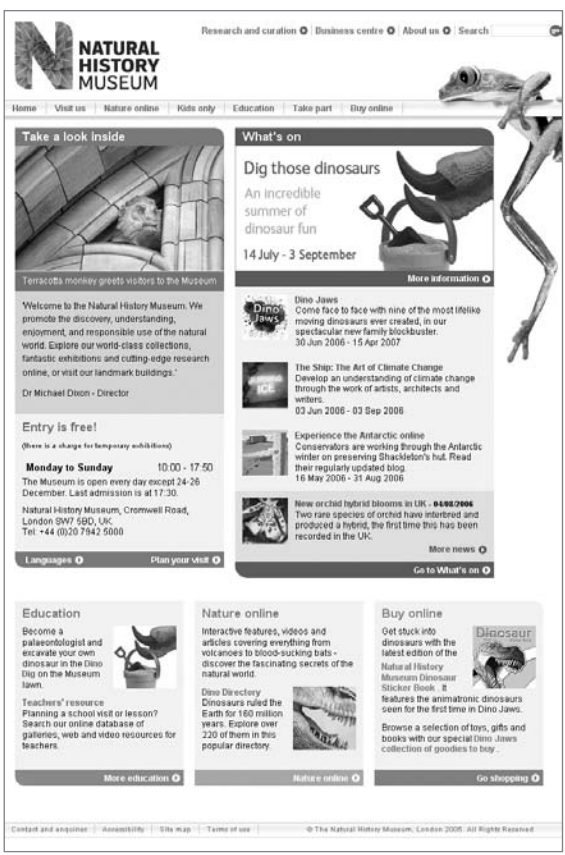

NATURAL HISTORY MUSEUM
Research and curation | Business centre | About us | Search
Home | Visit us | Nature online | Kids only | Education | Take part | Buy online
Take a look inside
Terracotta monkey greets visitors to the Museum
'Welcome to the Natural History Museum. We promote the discovery, understanding, enjoyment, and responsible use of the natural world. Explore our world-class collections, fantastic exhibitions and cutting-edge research online, or visit our landmark buildings.'
Dr Michael Dixon - Director
Entry is free!
Monday to Sunday 10:00 - 17:50
The Museum is open every day except 24-26 December. Last admission is at 17:30.
Natural History Museum, Cromwell Road, London SW7 5BD, UK
Tel: +44 (0)20 7942 5000
Languages
Plan your visit
What's on
Dig those dinosaurs
An incredible summer of dinosaur fun
14 July - 3 September
More information
Dino Jaws
Come face to face with nine of the most lifelike moving dinosaurs ever created, in our spectacular new family blockbuster.
30 Jun 2006 - 15 Apr 2007
The Ship: The Art of Climate Change
Develop an understanding of climate change through the work of artists, architects and writers.
03 Jun 2006 - 03 Sep 2006
Experience the Antarctic online
Conservators are working through the Antarctic winter on preserving Shackleton's hut. Read their regularly updated blog.
16 May 2006 - 31 Aug 2006
New orchid hybrid blooms in UK - 04/08/2006
Two rare species of orchid have interbred and produced a hybrid, the first time this has been recorded in the UK.
More news
Go to What's on
Education
Become a palaeontologist and excavate your own dinosaur in the Dino Dig on the Museum lawn.
Teachers' resource
Planning a school visit or lesson? Search our online database of galleries, web and video resources for teachers.
More education
Nature online
Interactive features, videos and articles covering everything from volcanoes to blood-sucking bats - discover the fascinating secrets of the natural world.
Dino Directory
Dinosaurs ruled the Earth for 160 million years. Explore over 220 of them in this popular directory.
Nature online
Buy online
Get stuck into dinosaurs with the latest edition of the Natural History Museum Dinosaur Sticker Book. It features the animatronic dinosaurs seen for the first time in Dino Jaws.
Browse a selection of toys, gifts and books with our special Dino Jaws collection of goodies to buy.
Go shopping
Contact and enquiries | Accessibility | Site map | Terms of use
© The Natural History Museum, London 2006. All Rights Reserved

FIGURE 3.1 The illustrations in this chapter are taken from London's Natural History Museum website (*www.nhm.ac.uk*).

It's really not very difficult to create a hypertext link; in fact, we're only talking about one HTML tag here. There's a lot to take into account with this independent little fella, though, from the way links are created, to what they can link to, to the countless options you have in styling them (whilst remembering that some restraint is needed to keep them as user-friendly as possible), to the area of accessibility, which has particular importance when it comes to links.

Anchor Elements and Hypertext References

So what tag do we use? `link`? Nope. That's for something else (see Chapter 1, "Getting Started"). Why, it's the diminutive `a` tag, of course! That's right: `a` for *anchor*.

What?!

We'll come to the reasoning behind this in a minute, but the most important part of an `a` element is actually the `href` attribute (meaning "hypertext reference"). The value of this attribute specifies the target of the link—where the browser should navigate to when the link is selected.

```
<a href="apage.html">A page</a>
```

URLS

A Universal Resource Locator is the location you want to link to (in the case of links, the value of the `href` attribute), be it a website, page, or any other file. The form of the URL can be different, depending on where the target is located—on the same page (such as "#something"—see "Page Anchors," below); in the same folder (such as "something.html"); the same site (such as "afolder/something.jpg" or "../alowerfolder/something.html"); or on a different server entirely (such as "http://www.htmldog.com/guides/").

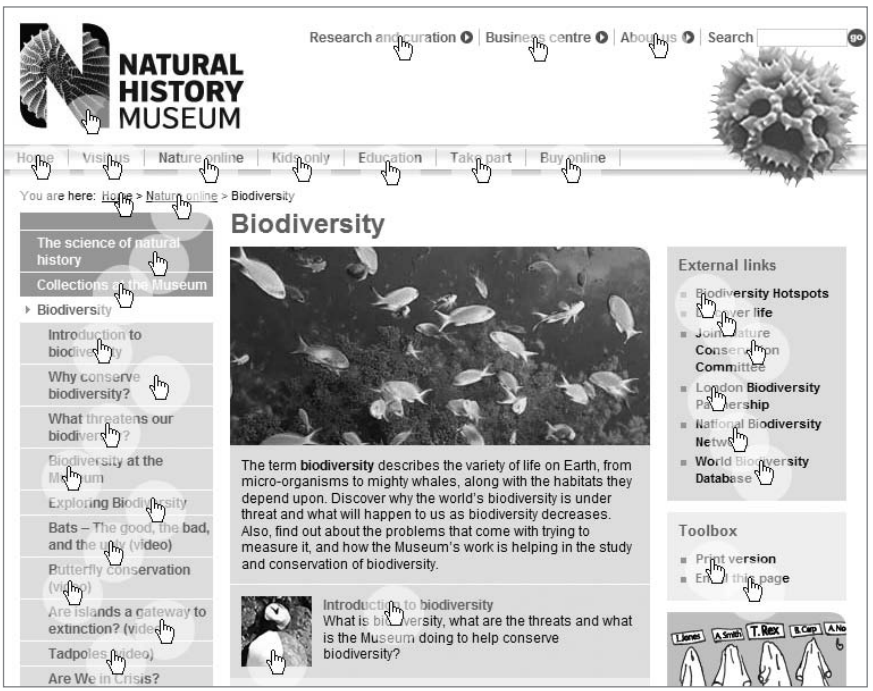

NATURAL HISTORY MUSEUM
Research and curation
Business centre
About us
Search
Home
Visit us
Nature online
Kids only
Education
Take part
Buy online
You are here: Home > Nature online > Biodiversity
The science of natural history
Collections at the Museum
Biodiversity
Introduction to biodiversity
Why conserve biodiversity?
What threatens our biodiversity?
Biodiversity at the Museum
Exploring Biodiversity
Bats – The good, the bad, and the ugly (video)
Butterfly conservation (video)
Are islands a gateway to extinction? (video)
Tadpoles (video)
Are We in Crisis?
Biodiversity
The term biodiversity describes the variety of life on Earth, from micro-organisms to mighty whales, along with the habitats they depend upon. Discover why the world's biodiversity is under threat and what will happen to us as biodiversity decreases. Also, find out about the problems that come with trying to measure it, and how the Museum's work is helping in the study and conservation of biodiversity.
Introduction to biodiversity
What is biodiversity, what are the threats and what is the Museum doing to help conserve biodiversity?
External links
Biodiversity Hotspots
Join Nature Conservation Committee
London Biodiversity Partnership
National Biodiversity Network
World Biodiversity Database
Toolbox
Print version
T. Rex

FIGURE 3.2 A whole bunch o' links, a whole bunch o' `a` elements.

Page Anchors

OK, so here's the reason why we're using a tag called "anchor." In the olden days, back when giant reptiles were plodding around and all mammals were the size of shrews, "page anchors"—the points in a page that can be jumped to from selecting a link—were defined with the `a` element.

Nowadays, such an explicit page anchor element isn't needed because any element with an `id` attribute (see Chapter 1) can act as an anchor.

Linking to a completely different page with something like `href="something.html"` is very straightforward, but jumping down (or up) a page to a page anchor is just as simple. To refer to a page anchor, you simply put a number sign—#—before the name of the id you want to jump to. The label that follows the # character is referred to as the "fragment." So in the following example, selecting the "nitty-gritty" link

(with the attribute `href="#nittygritty"`) will cause the browser to jump down the page to the "Nitty-Gritty" `h2` element (with the attribute `id="nittygritty"`):

```
<p>Jump straight to the <a href="#nittygritty">nitty-gritty</a>
<!--[A whole load of content]-->
<h2 id="nittygritty">Nitty-Gritty</h2>
```

www.htmldog.com/examples/target.html

You can also jump to an anchor in another page by simply bolting on the `"#whatever"` to the end of the URL. So to jump to the above "Nitty-Gritty" element from another page, you would use `href="whatever.html#nitty-gritty"`.

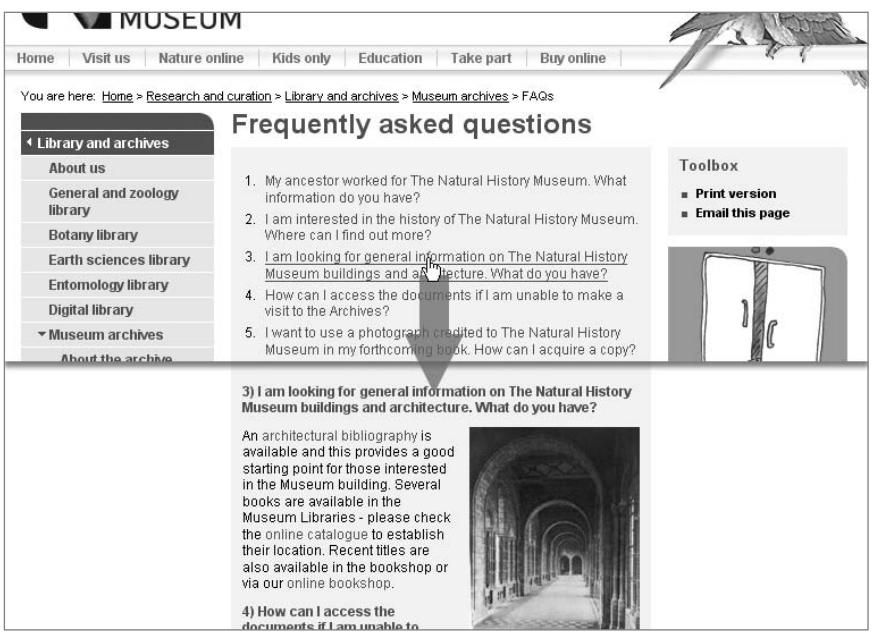

MUSEUM
Home
Visit us
Nature online
Kids only
Education
Take part
Buy online
You are here: Home > Research and curation > Library and archives > Museum archives > FAQs
Library and archives
About us
General and zoology library
Botany library
Earth sciences library
Entomology library
Digital library
Museum archives
Frequently asked questions
1. My ancestor worked for The Natural History Museum. What information do you have?
2. I am interested in the history of The Natural History Museum. Where can I find out more?
3. I am looking for general information on The Natural History Museum buildings and architecture. What do you have?
4. How can I access the documents if I am unable to make a visit to the Archives?
5. I want to use a photograph credited to The Natural History Museum in my forthcoming book. How can I acquire a copy?
Toolbox
Print version
Email this page
3) I am looking for general information on The Natural History Museum buildings and architecture. What do you have?
An architectural bibliography is available and this provides a good starting point for those interested in the Museum building. Several books are available in the Museum Libraries - please check the online catalogue to establish their location. Recent titles are also available in the bookshop or via our online bookshop.
4) How can I access the

FIGURE 3.3 FAQ pages commonly use page anchors—a link at the top of the page scrolls the browser to a heading farther down the page.

Link States: Link, Visited, Hover, Focus, and Active

There are five link states that become particularly prevalent when using CSS:

1. One for when a link has not yet been visited
2. One for when a link has been visited
3. One for when a link is being hovered over
4. One for when a link receives focus (for example, when it is tabbed to by keyboard input)
5. One for when a link is being selected.

By default, browsers tend to render unvisited links in blue, visited links in purple (or mushroom-soup-sick brown in IE) and active links in red, but you can style these, as well as the hover and focus states, however you choose.

By using the selector `a` on its own, you can set properties that will apply to all of the link states (such as `a { color: blue }`). This is a common practice, but one that should be followed up with styles for the individual link states to take advantage of the valuable cues they can provide to users. To change the properties of the link states independently, you can use the pseudo-classes `link` (for the default, unvisited state), `visited`, `hover`, `focus`, and `active`, as in the following example:

```
a:link { color: blue; }
a:visited { color: purple; }
a:hover { text-decoration: none; }
a:focus { background-color: yellow; }
a:active { color: red; }
```

Link State. A color that stands out from surrounding text is usually selected for a link, but to recognize the needs of colorblind users you should not rely on color alone. An underline text decoration is applied by default with the `a` element. If you decide to have links without underlines, consider adding another visual cue, such as displaying them in bold. Keep in mind, though, that underlines are still associated with links and make them more instantly recognizable than any other visual cue, particularly when they appear in the middle of content. (This is not as important for links in a well-defined navigation area, for example.)

Visited State. The visited state is notoriously underused. By making use of the `visited` pseudo-class to make links look slightly different (in a lighter color, for example), you present a cue to let users know which pages they have already been to, and which pages they haven't—a distinction that has been shown in usability tests to help users make sense of a site.

Hover State. The `hover` state, which comes into play when the cursor moves over a link, has also been shown to be useful to users.

Possible and popular combinations include turning off the underline on links using `a:link { text-decoration: none; }` and then having it appear when the link is hovered over with `a:hover { text-decoration: underline; }`.

You can obviously change colors, too, but it isn't usually wise to change the size, font-weight, or font-style of a link when it is hovered over because it can push out surrounding content (what with it taking up more room than it did in its pre-hover state).

So a basic hover effect you would use might look something like this:

```
a {
    color: #900;
    font-weight: bold;
    text-decoration: none;
}
a:visited {
    color: #c00;
}
a:hover {
    color: #900;
    text-decoration: underline;
}
```

The initial, unvisited link will have no underline, but will be bold to make it stand out from surrounding text. Visited links will be slightly lighter to differentiate them from links that have not been selected. And when the link is hovered over, an underline will appear.

Focus State. The `focus` pseudo-class, which—although it could provide a good visual indicator for people tabbing through links—is actually the least helpful of

these pseudo-classes because it isn't supported by Internet Explorer (although its effects can be mimicked with JavaScript—see *www.htmldog.com/articles/suckerfish/focus/*).

Active State. Finally, the `active` pseudo-class can be used to style a link that is being selected (as in, being clicked), for that extra bit of user feedback.

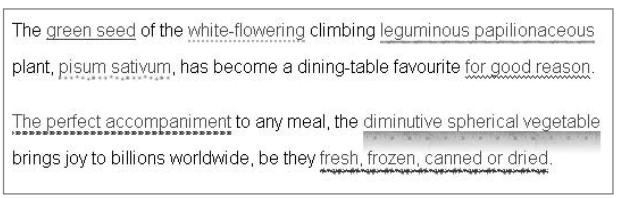

The green seed of the white-flowering climbing leguminous papilionaceous plant, pisum sativum, has become a dining-table favourite for good reason.
The perfect accompaniment to any meal, the diminutive spherical vegetable brings joy to billions worldwide, be they fresh, frozen, canned or dried.

FIGURE 3.4 OK, so underlining links with text-decoration is quite common. But you could be a bit fancier and apply borders or even background images to achieve a more customized result. Take a read of the short article at *www.htmldog.com/articles/underlines/* or just jump in and take a look at the bare-bones example:

www.htmldog.com/examples/underlines.html

Accessible Links

Links are an essential tool of navigation—without them, the Web would be a very difficult place to get around. For that reason, it's important to pay special attention to the accessibility of links to make sure everyone can use web pages effectively, even if they don't use pointing devices or if they rely on nonvisual browsers. This section covers some techniques for addressing those considerations.

Tabbing

Those who do not or cannot use a pointing device such as a mouse will often "tab" through links with a keyboard or other input device, whereby the user can cycle through the links with every press of a key (such as the Tab key). One of the benefits of building web pages according to web standards is that the content tends to fall into a logical order, as do any links within it, so a good tab order is usually automatic.

To explicitly set the tab order of links on a page, you can use the `tabindex` attribute. Numerical values dictate what links will come where in the order, `1` being first, `2` being second, etc.

```
<a href="wherever.html" tabindex="1">Wherever</a>
```

You may want to do this if you believe some links are of greater importance or interest to the user than others and so want to make the process of focusing in on those links quicker.

Access Keys

Links can also be accessed by keyboard shortcuts, which remove the need for tabbing through a large number of links before reaching the desired link. These shortcuts can be applied using the `accesskey` attribute.

```
<a href="whatever.html" accesskey="w">Whatever</a>
```

When a user presses the access key, plus Ctrl or Alt (depending on the platform), the browser will move focus to the link assigned to that access key.

It isn't always necessary—or even realistically possible—to add access keys to all links on a page, but it is suggested that they should be added to major links, such as primary navigation.

THE (BIG) PROBLEM(S) WITH ACCESS KEYS

There are two major problems with access keys.

The first is that although some screen-readers will read out access key values, there is no good way of letting the user know visually what the access keys are. As there is no universally accepted standard, websites tend to use different access keys to accomplish different tasks in different ways.

Suggestions for conveying the access key information have included explicitly stating what the access key is (such as `<a href="blah.html" accesskey="b">Blah (B)</a>`) or using CSS to underline or in some other way highlight the letter of the corresponding access key within the link text. The problem with this approach is that you cannot expect a user to know what the hell these things indicate. Most people won't want to use access keys or even care what they are let alone know what they are. Ideally, the browser would somehow convey access keys independently of the web page view itself, but there's nothing the web designer can do about this.

Another method of conveying access key information is to have an accessibility statement web page that explicitly states what they are and what links they are assigned to. The downside of this is that users have to navigate to another page before reading the page they are really interested in. This is something that users notoriously rarely do.

If you choose to use access keys despite this problem, there is a *second* problem waiting in the shadows to spoil the fun. This problem is that access keys can cause conflicts with browser shortcuts. A user may want to select the File menu from the browser window menu by pressing Alt+F on the keyboard, but find that something quite unexpected happens because F is assigned to a link on the web page and steals the limelight.

It is extremely difficult to predict what, if any, keyboard shortcuts are safe to use as access keys. You can work out what shortcuts apply to your browser, even to a number of browsers, but what about browsers in different languages? The trouble is that if you want your web page to appeal to an international audience (this is the *Inter*net we're talking about, after all) then all of those browser menu options you are used to will have different wording and with it different keyboard shortcuts. This means that pretty much any letter assigned to an access key will conflict with a browser shortcut somewhere, causing confusion, hindering usability, and defeating the whole object of the access key.

So what to do? Firstly, due to the unfortunate problems, don't worry too much about them—they're just not very practical. If you do choose to use them, however, try sticking to numeric access keys and go with that rarely visited accessibility statement (such as the one in Figure 3.5).

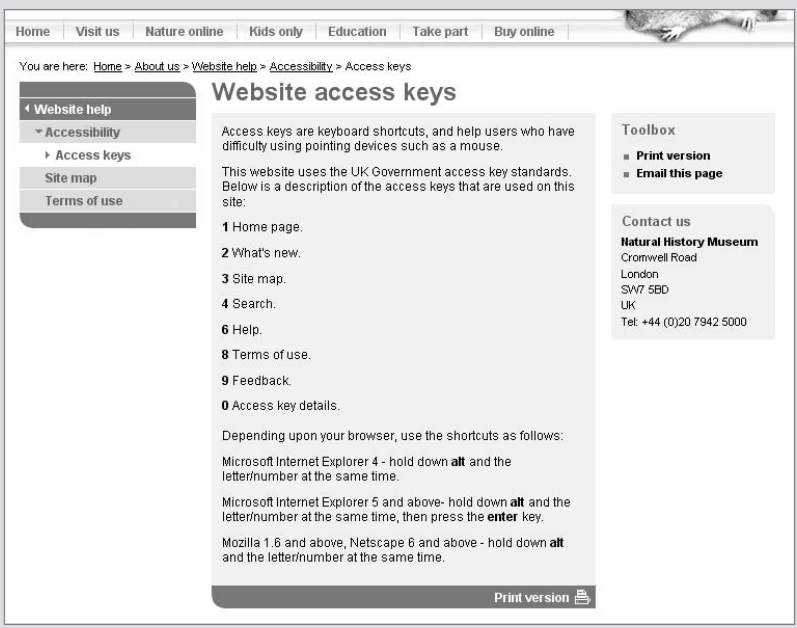

Home
Visit us
Nature online
Kids only
Education
Take part
Buy online
You are here: Home > About us > Website help > Accessibility > Access keys
Website help
Accessibility
Access keys
Site map
Terms of use
Website access keys
Access keys are keyboard shortcuts, and help users who have difficulty using pointing devices such as a mouse.
This website uses the UK Government access key standards. Below is a description of the access keys that are used on this site:
1 Home page.
2 What's new.
3 Site map.
4 Search.
6 Help.
8 Terms of use.
9 Feedback.
0 Access key details.
Depending upon your browser, use the shortcuts as follows:
Microsoft Internet Explorer 4 - hold down alt and the letter/number at the same time.
Microsoft Internet Explorer 5 and above- hold down alt and the letter/number at the same time, then press the enter key.
Mozilla 1.6 and above, Netscape 6 and above - hold down alt and the letter/number at the same time.
Print version
Toolbox
Print version
Email this page
Contact us
Natural History Museum
Cromwell Road
London
SW7 5BD
UK
Tel: +44 (0)20 7942 5000

FIGURE 3.5 Like many sites that have access keys, the NHM site has a page explaining what they are and where they will take the user. Note that the site has also chosen numeric access keys to minimize problems with browser shortcuts.

Link Titles

As with many tags, you can use the common attribute `title` within the opening *a* tag to provide more information about a link, such as a description of where the link will take the user.

Although the text making up a link should describe the target, if it doesn't (if you are forced to use "click here" or "more" as the link text, for example), then the title

can be used to provide more information. Don't assume this fixes the problem of crap link text, though—a user's screen-reader isn't necessarily going to be set up to read link titles.

Pop-ups

Pop-ups are a serious accessibility no-no because they unexpectedly navigate the user to a new window and break the "back button" functionality. If you find yourself in the position of using them, however, there are a few steps you can take to make their application more accessible.

Firstly, the link title should be used to state that the link will cause a new window to pop up.

Secondly, the value of the `href` attribute should be the page used in the pop-up and the JavaScript used to launch the pop-up should return false. This way, if a user does not have JavaScript or has it disabled, the content of the pop-up will be navigated to, just not in a pop-up.

```
<a href="eviltroll.html" title="Launch Evil Troll in a little
pop-up" onclick="popup(this); return false;">Evil Troll</a>
```

Contrary to popular belief, you do not need to replicate the `onclick` event with the `onkeypress` event, although this would seem to make sense in accommodating keyboard input as well as "clicking" input. This is because as well as being invoked by clicking, `onclick` will also be invoked by key-pressing.

Adjacent Links

Another issue raised by the Web Accessibility Initiative (WAI) is that adjacent links should be separated by more than just spaces so that they can be discerned by screen-readers.

There is argument over the continued validity of this point due to advances in screen-reader technology that allow links to be read more clearly, but a problem still remains with adjacent links, and they should be avoided if at all possible.

Separating navigation links with spaces is rarely anything more than presentational, and smacks of the bad practice of using a number of   (nonbreaking space)

characters for quick-fix presentation. It is bad grammar to have a series of unbroken, unrelated words (or abbreviations) in a paragraph.

This case usually calls for such items to be placed in a list (see Chapter 6, "Lists"), but sometimes situations may arise where a paragraph is preferred.

For example, if you had a number of links claiming standards compliance, you might want "XHTML CSS WAI," but if this were placed in a paragraph it should be punctuated with commas such as "XHTML, CSS, WAI" or at least with some separator, such as "XHTML | CSS | WAI."

It is not entirely uncommon (particularly in blogs) to find adjacent words in a phrase linking to different places (such as `<a>as</a> <a>some</a> <a>have</a> <a>said</a>`). The trouble with this is that the link text rarely describes where it is taking the user.

Skipping Navigation

When a sighted user is presented with a page, it doesn't take that long to focus on the content—he or she doesn't need to read all of the navigation options. Those who rely on screen-readers don't necessarily have this luxury. A screen-reader will read through the entire content of a web page in the order it appears in the HTML, which often means large portions of navigational elements being read out on every page before it gets to the intended informative content.

To take this situation into account, you can create a link that will give the user the option to skip the navigation and jump straight into the content.

```
<p class="accessaid"><a href="#content">Skip to content</a></p>
<!--[Big chunk of navigation]-->
<div id="content">
    <!--[The nitty gritty]-->
</div>
```

This code should look familiar—it is simply a basic link to a page anchor (defined by the ID "content"). Not only would the browser jump down the page when the "Skip to content" link is selected, the point at which a screen-reader continues to read would also jump to that point, so bypassing the reading of the navigation.

Although there may be users (such as those with motor disabilities) who exploit visuals but do not use pointing devices to whom this skip-to-content technique would be beneficial, it is still a good idea to hide this link from view using CSS to avoid confusion (the link won't seem to do anything to the vast majority of users, who won't need to use it):

```
.accessaid {
    position: absolute;
    height: 0;
    overflow: hidden;
}
```

(This method of hiding, and the properties used in the example, are described in Chapter 5, "Layout.")

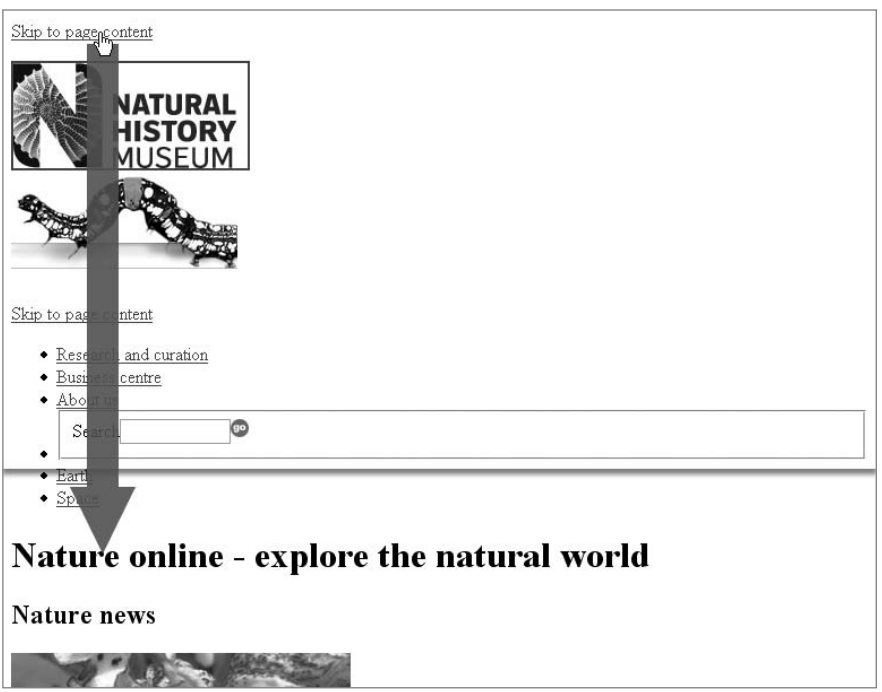


FIGURE 3.6 When the CSS is turned off, the Skip navigation links are revealed.

On the flip side of this, a page could be structured whereby the content comes before the navigation options. In this case, you can just as easily add a "Skip to navigation" option for users who might want to access the navigation without having to read through all of the content.

chapter 4

Images

IMAGES ARE PERHAPS the most obvious way to add visual appeal to a web page. If you want anything more than the most minimal of minimal sites you'll want to use images in one way or another, whether it's showing off your holiday photos or adding curved corners to your layout. Images also used to be used all over the place to help lay out pages as invisible "spacer GIFs," but now that we've got CSS layout on our side (see Chapter 5, "Layout"), we can do away with those crazy days.

You can split the use of images quite neatly into two camps: one for content (such as those holiday photos) and one for presentation (that'll be those curved corners), and unless you've just opened the book for the first time and landed on this page, you should know by now that HTML should be used for the former and CSS for the latter.

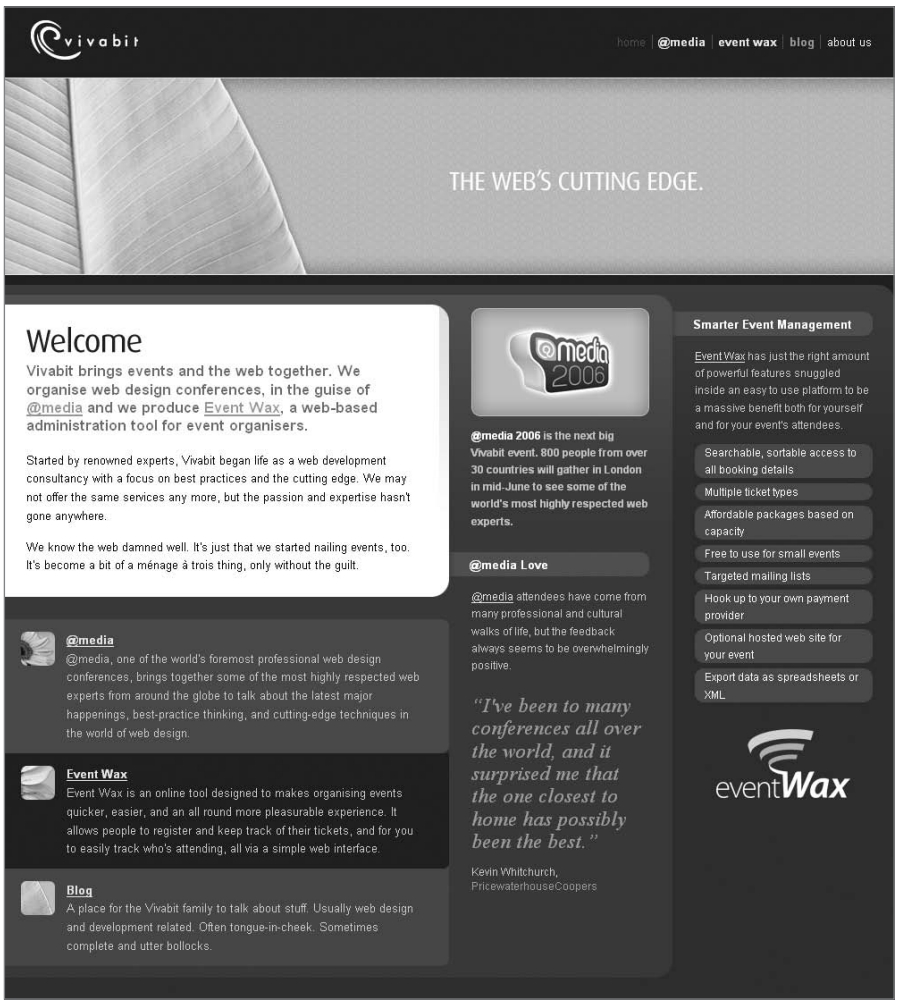

vivabit
home | @media | event wax | blog | about us
THE WEB'S CUTTING EDGE.
Welcome
Vivabit brings events and the web together. We organise web design conferences, in the guise of @media and we produce Event Wax, a web-based administration tool for event organisers.
Started by renowned experts, Vivabit began life as a web development consultancy with a focus on best practices and the cutting edge. We may not offer the same services any more, but the passion and expertise hasn't gone anywhere.
We know the web damned well. It's just that we started nailing events, too. It's become a bit of a ménage à trois thing, only without the guilt.
@media
@media, one of the world's foremost professional web design conferences, brings together some of the most highly respected web experts from around the globe to talk about the latest major happenings, best-practice thinking, and cutting-edge techniques in the world of web design.
Event Wax
Event Wax is an online tool designed to makes organising events quicker, easier, and an all round more pleasurable experience. It allows people to register and keep track of their tickets, and for you to easily track who's attending, all via a simple web interface.
Blog
A place for the Vivabit family to talk about stuff. Usually web design and development related. Often tongue-in-cheek. Sometimes complete and utter bollocks.
@media 2006
@media 2006 is the next big Vivabit event. 800 people from over 30 countries will gather in London in mid-June to see some of the world's most highly respected web experts.
@media Love
@media attendees have come from many professional and cultural walks of life, but the feedback always seems to be overwhelmingly positive.
"I've been to many conferences all over the world, and it surprised me that the one closest to home has possibly been the best."
Kevin Whitchurch, PricewaterhouseCoopers
Smarter Event Management
Event Wax has just the right amount of powerful features snuggled inside an easy to use platform to be a massive benefit both for yourself and for your event's attendees.
Searchable, sortable access to all booking details
Multiple ticket types
Affordable packages based on capacity
Free to use for small events
Targeted mailing lists
Hook up to your own payment provider
Optional hosted web site for your event
Export data as spreadsheets or XML
eventWax

FIGURE 4.1 The illustrations in this chapter are taken from the Vivabit website (*vivabit.com*).

The `img` Element

The `img` element allows you to plonk an image straight into your HTML.

```
<img src="images/sifaka.jpg" alt="Leaping sifaka" />
```

The required `src` attribute points to the location of the image file.

The `alt` attribute, which is also required, is for specifying alternative text. It serves an important accessibility task: It provides an "alternative" to the image for those who cannot see the image itself (such as those reliant on screen-readers). As an added bonus, most browsers use this attribute to provide placeholder text while the image is downloading. The value can give an idea of what the image represents, but doesn't have to—it can be anything that would adequately serve as alternative content to the image.

www.htmldog.com/examples/images1.html

You can also use the `longdesc` attribute, the value of which would be the location (in the form of a URL) of a description of the image. The idea behind this is that when there is a very detailed image (such as a map or a chart) that may need a solid, long explanation, you don't necessarily want to bog down the page with massive `alt` attributes (which should be short and sweet). `longdesc` gives the user an option to navigate to a page that will explain what is going on.

BORDER ANNIHILATION

Note that if you include an `img` element inside a link, browsers will tend to apply a border to the image by default. You can easily annihilate the border with CSS—`img { border: 0; }`. See Chapter 3 for more on links and Chapter 5 for more on borders.

vivabit
home | @media | event wax | blog | about us
ABOUT US
The Web's Cutting Edge
Web events company Vivabit organises the renowned @media conference, and is the maker of Event Wax, the web-based event management application.
Started by experienced web professionals, Vivabit began life as a web design and development consultancy, gradually concentrating more and more on the events side of our services, until events became our sole focus.
What Floats Our Boat
Technical excellence, being genuinely cutting-edge, applying the latest techniques and working with the best in the business.
Standards compliance, following best technological practices for best results.
User centric web design, because the happier the user, the more successful the web site.
Good communication being just as important as technical expertise.
High levels of accessibility without compromising graphic design.
Having fun and enjoying what we do.
We could rant on about the organisations we've been proud to work with, the world-class talent that works for Vivabit or maybe even our argumentative take on the state of accessibility, but we won't, because you've most likely got the general idea. If there's anything you want to know about us, just ask. We probably won't bite.
The Players
We're a small team, and proud of it. From a core base of technical experts, we tap into a close network of world-leading specialists whenever necessary.
Patrick Griffiths
Some years ago Patrick decided to put all of his eggs in one basket and concentrate on front-end web development. Since then he has worked on websites for Vodafone, Wiley and local government and has established himself as a leader in his field with several well received professional articles and a book to his name.
He's a film lover and a music man, but rates Charles Darwin and Valentino Rossi as history's two greatest figures.
Dan Webb
A web application development nut, Dan has demonstrated his considerable skills working on projects for the likes of Nike, Sainsbury's, Dulux, Nestle and various educational and governmental institutions and in a past life was a senior developer for über-agency AKQA.
On those occasions he's not programming, he enjoys hip-hop and cheese slices. Although not necessarily at the same time.
Oh, and yes, that is his real name.
Vivabit Soho!
We're damned excited to be situated in the infamous Soho district of central London's West End.
Broadwick Street's previous claim to fame (before we moved there) was that it was the source of the cholera epidemic of 1854.
This fact was discovered by John Snow, after whom the pub directly opposite our office is named.

FIGURE 4.2 Even in a graphically rich web page, img elements tend to be few and far between. Take a sub-page from this website, for example…

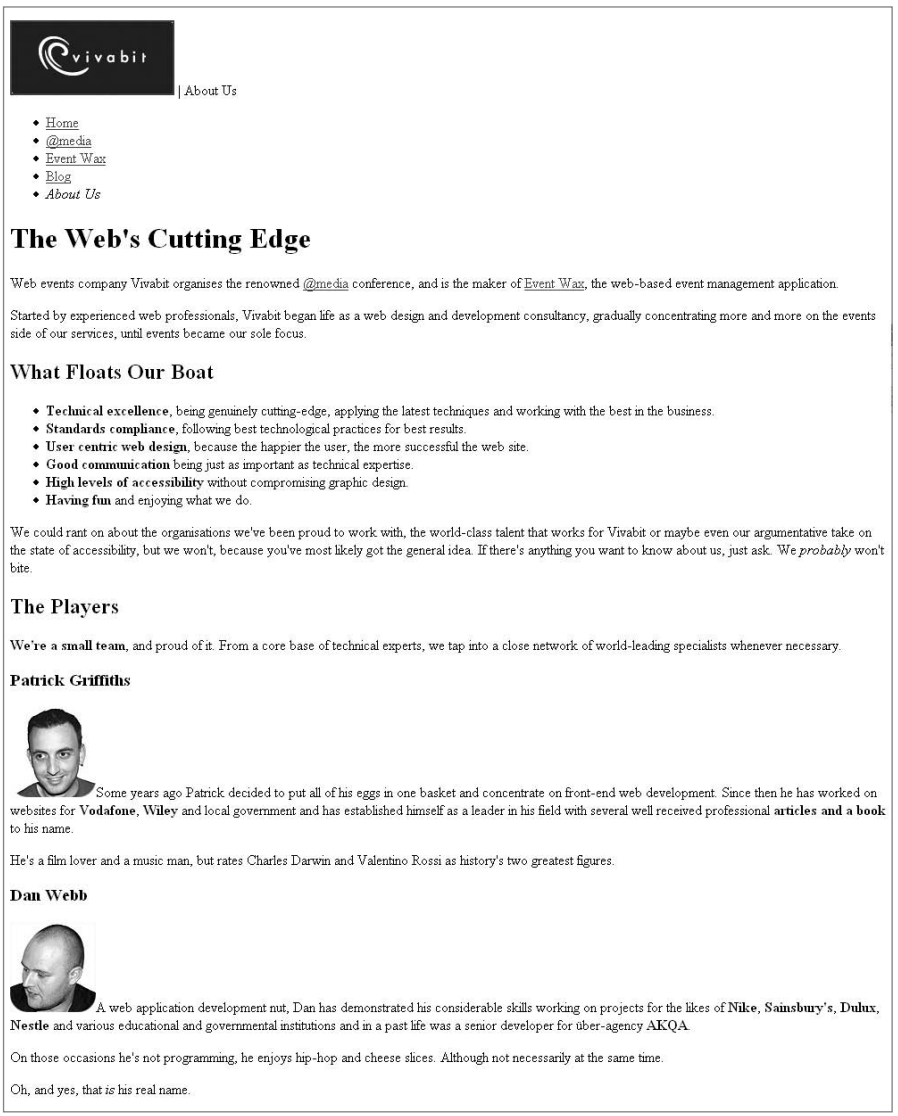

vivabit
| About Us
Home
@media
Event Wax
Blog
About Us
The Web's Cutting Edge
Web events company Vivabit organises the renowned @media conference, and is the maker of Event Wax, the web-based event management application.
Started by experienced web professionals, Vivabit began life as a web design and development consultancy, gradually concentrating more and more on the events side of our services, until events became our sole focus.
What Floats Our Boat
Technical excellence, being genuinely cutting-edge, applying the latest techniques and working with the best in the business.
Standards compliance, following best technological practices for best results.
User centric web design, because the happier the user, the more successful the web site.
Good communication being just as important as technical expertise.
High levels of accessibility without compromising graphic design.
Having fun and enjoying what we do.
We could rant on about the organisations we've been proud to work with, the world-class talent that works for Vivabit or maybe even our argumentative take on the state of accessibility, but we won't, because you've most likely got the general idea. If there's anything you want to know about us, just ask. We probably won't bite.
The Players
We're a small team, and proud of it. From a core base of technical experts, we tap into a close network of world-leading specialists whenever necessary.
Patrick Griffiths
Some years ago Patrick decided to put all of his eggs in one basket and concentrate on front-end web development. Since then he has worked on websites for Vodafone, Wiley and local government and has established himself as a leader in his field with several well received professional articles and a book to his name.
He's a film lover and a music man, but rates Charles Darwin and Valentino Rossi as history's two greatest figures.
Dan Webb
A web application development nut, Dan has demonstrated his considerable skills working on projects for the likes of Nike, Sainsbury's, Dulux, Nestle and various educational and governmental institutions and in a past life was a senior developer for über-agency AKQA.
On those occasions he's not programming, he enjoys hip-hop and cheese slices. Although not necessarily at the same time.
Oh, and yes, that is his real name.

FIGURE 4.3 The only `img` elements in the HTML are the logo and the headshots.

WIDTH? HEIGHT? IN HTML?

The `width` and `height` attributes can be quite useful. They let the browser know how much space to reserve on a page even before the image itself starts to download. Without this information, the browser will only know the image size once the image starts to download, which can mean that it needs to redraw the page, causing surrounding content to jump all over the place. For example, if there is an `img` element without width and height settings in a page surrounded by a whole load of text, the browser will render the text first, leaving a small default area for the image. When it comes to download the image and realizes that it is actually bigger than the space it left, it will need to readjust where the text flows to accommodate the image.

```
<img src="images/sifaka.jpg" alt="Leaping sifaka" width="500"
height="129" />
```

But *hold* on a minute! Width and height are spatial concepts—completely presentational; shouldn't this be something that is done with CSS? Well, ideally, yes, with the `width` and `height` properties (`width: 500px; height: 129px;` for example—see Chapter 5). But this approach isn't always practical. If you had a lot of different `img` elements on a page (or across a whole site, for that matter) and they all had different dimensions (if it was some kind of online photo album, for example), then you would need to create classes for every image (see Chapter 1, "Getting Started"), which could lead to an unwieldy amount of CSS. In such a case, although it messes with the whole structure/presentation philosophy, the `width` and `height` attributes might be the most practical route to take.

JPEG, GIF, OR PNG?

This is a little outside the realms of HTML and CSS, but because it is so important in the construction of a web page with images it is worth briefly noting the different image file formats for any beginners out there.

JPEG, GIF, and PNG use compression algorithms to deliver lightweight images for web pages. These algorithms work in different ways and each is suitable for different situations. Most decent image manipulation software programs will give you some control over the degree of compression, allowing you to strike a balance between quality and file size (the more compressed an image, the smaller the file size and the lower the quality).

JPEGs should normally be used for detailed images such as photographs.

GIFs should be used for images with solid blocks of color. This format allows up to 256 colors, including transparency. The fewer the colors in an image, the smaller the file size.

PNGs achieve a similar result to GIFs but in a more sophisticated way. They allow more colors and also "alpha" transparency, which means individual pixels can be set to a certain degree of transparency (ranging from opaque to completely transparent). Unfortunately, the alpha transparency in PNGs is not supported in the commonly used Internet Explorer 6, although IE7 will be more well behaved.

Image Maps

Let's keep this brief: Image maps, which allow a user to click on various parts within an image, are not widely used (certainly not in a good way, anyway) and there are usually better alternatives.

There are two flavors: server-side image maps, which belong in Satan's toolbox and are discussed in Chapter 9, "Forms," and client-side image maps, which are cobbled together with the map and area elements.

```
<img src="atlast.gif" alt="Atlas" usemap="atlas" />
<map name="atlas" id="atlas">
    <area shape="rect" coords="0,0,115,90" href="northamerica.html"
alt="North America" />
    <area shape="poly" coords="113,39,187,21,180,72,141,77,117,86"
href="europe.hmtl" alt="Europe" />
    <area shape="poly" coords="119,80,162,82,175,102,183,102,175,148
,122,146" href="africa.html" alt="Africa" />
</map>
```

In this example, the `img` element is the image. The `map` element then links onto that image via the `usemap` attribute in the `img` element matching the `name` attribute in the `map` element. Each `area` element then defines an area on the image (with a shape and coordinates) and provides a link. So if this were a map of the world, then you could make each continent clickable.

Why aren't these much use? Because there aren't many valid applications for them (geographical maps are the most obvious use), and even when you have a valid use (splitting one big image into navigational links, a popular crime of the past, is not a valid use) they're not very user friendly because it's not immediately obvious that the image is a clickable map. They may seem clever, but they're perhaps too clever for their own good.

Background Images

Because images are so often used in a purely presentational capacity, rather than as genuine content, CSS is usually preferable to HTML for dealing with them. `img` elements used to be prolific—plastered any and everywhere to achieve even the slightest presentational effect (and are still commonly used as such today). But now, in the web standards era, the image niche is dominated by another, slicker animal—the CSS background image.

The `background-image` property can be used to specify an image to be used as a background for just about any element box—from the page body to a paragraph to a link. Use it on its own, and the image will magically tile itself across the background of the element starting from the top left corner and repeating horizontally and vertically, filling the box.

```
body { background-image: url(images/sifakabg.gif); }
```

www.htmldog.com/examples/images2.html

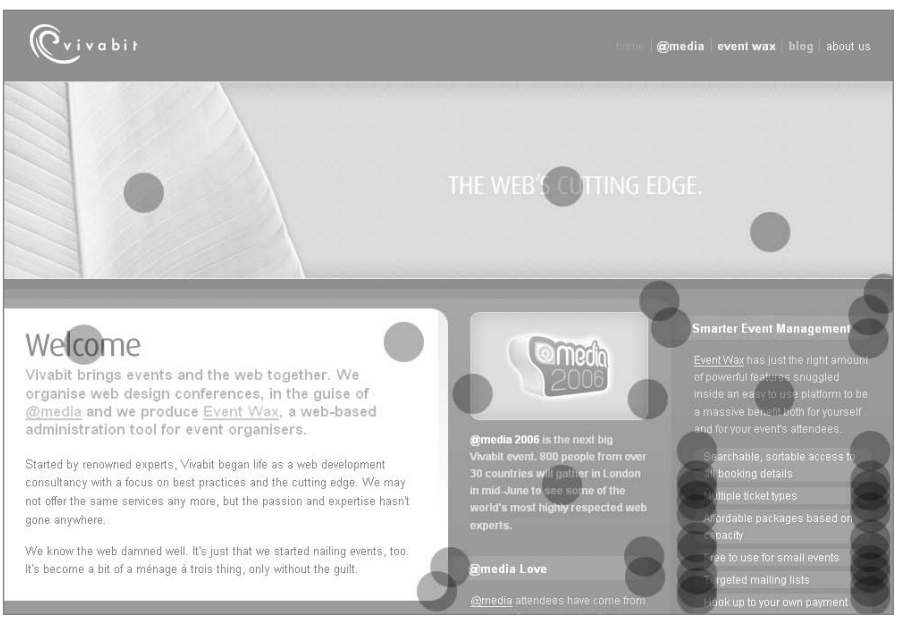

vivabit
home | @media | event wax | blog | about us
THE WEB'S CUTTING EDGE.
Welcome
Vivabit brings events and the web together. We organise web design conferences, in the guise of @media and we produce Event Wax, a web-based administration tool for event organisers.
Started by renowned experts, Vivabit began life as a web development consultancy with a focus on best practices and the cutting edge. We may not offer the same services any more, but the passion and expertise hasn't gone anywhere.
We know the web damned well. It's just that we started nailing events, too. It's become a bit of a ménage à trois thing, only without the guilt.
@media 2006
@media 2006 is the next big Vivabit event. 800 people from over 30 countries will gather in London in mid-June to see some of the world's most highly respected web experts.
@media Love
@media attendees have come from
Smarter Event Management
Event Wax has just the right amount of powerful features snuggled inside an easy to use platform to be a massive benefit both for yourself and for your event's attendees.
Searchable, sortable access to all booking details
Multiple ticket types
Affordable packages based on capacity
Free to use for small events
Targeted mailing lists
Hook up to your own payment

FIGURE 4.4 Spot the background images. They're all over the place—15 in this screenshot alone.

You can control aspects of the background image with the `background-attachment`, `background-repeat`, and `background-position` CSS properties.

`background-attachment` determines whether the background image should scroll with the content of a box. It can be used to specify whether the image should scroll

with the rest of the page (which it normally would do) or whether it should be fixed to the viewport (the viewing area of the browser window, rather than the page).

```
body {
    background-image: url(images/sifakabg.gif);
    background-attachment: fixed;
}
```

This example will plaster the "sifakabg.gif" image across the page, and, rather than the pattern scrolling as it would do on a long page with lots of content, it will stick right where it is, with the rest of the page scrolling over the top.

You don't have to have the background image tiled (repeated over and over, horizontally and vertically as space allows). By using the `background-repeat` property you can decide whether you want it to repeat just horizontally (`repeat-x`), just vertically (`repeat-y`), or not at all (`no-repeat`).

```
body {
    background-image: url(images/sifakabg.gif);
    background-repeat: no-repeat;
}
```

www.htmldog.com/examples/images3.html

Those areas of the element that are not taken up by the background image will be transparent, unless coupled with a background color (see Chapter 1), which would paint the rest of the area that color.

Background images will start at the top left corner of a box by default, but you can change this with the `background-position` property, which is particularly useful when `background-repeat` is set to `no-repeat`, for example.

Values can be `top`, `right`, `bottom`, `left`, `center`, a length, a percentage, or a combination of these (such as `top left`).

```
body {
    background-image: url(images/sifakabg.gif);
    background-repeat: no-repeat;
    background-position: center;
}
```

FIGURE 4.5 The leaf image is set to `background-repeat: no-repeat` to achieve just one instance of it. The little spots that make up rest of the strip are one small tessellating image set to repeat.

Another one of those funky shorthand properties is `background`, which can combine some or all of `background-color` (which we came across in Chapter 2, "Text"), `background-image`, `background-repeat`, `background-attachment`, and `background-position` into one.

```
body { background: #0084c7 url(images/sifakabg.gif) top left fixed
no-repeat; }
```

Although all of the examples so far have been applying backgrounds to the `body` element box, you can apply them to any visible element on the page, be it a paragraph, a link, a table, or even a partially transparent `img` element, if you really want to.

TECHNIQUE: ROUNDED CORNERS

Background images aren't just about the bigger picture—they are used for every decorative effect. In Figure 4.6 two rounded corners are applied to a content area. The first is applied to the area's container and the last is applied to the bottom paragraph, so there is no need for any extra markup.

Welcome
Vivabit brings events and the web together. We organise web design conferences, in the guise of @media and we produce Event Wax, a web-based administration tool for event organisers.
Started by renowned experts, Vivabit began life as a web development consultancy with a focus on best practices and the cutting edge. We may not offer the same services any more, but the passion and expertise hasn't gone anywhere.
We know the web damned well. It's just that we started nailing events, too. It's become a bit of a ménage à trois thing, only without the guilt.

FIGURE 4.6 Two rounded corners are applied to a content area.

As long as you have enough elements to latch CSS onto, you can apply more than one background to a part of the page. For example, you could add one rounded corner to a paragraph by applying a background image to the top left of a `p` element, but if you had something like `<p><span><span><span>whatever </span></span</span></p>` then you could apply a rounded corner to each of

the elements (`p` for the top left corner, `p span` for the top right corner, `p span span` for the bottom left corner, and `p span span span` for the bottom right corner) using something like this:

```
p {
    background: white url(images/sifakaptl.gif) top left no-
repeat;
}
p span {
    background: url(images/sifakaptr.gif) top right no-repeat;
}
p span span {
    background: url(images/sifakapbr.gif) bottom right no-repeat;
}
p span span span {
    background: url(images/sifakapbl.gif) bottom left no-repeat;
}
```

The p element applies one of the corner images (top left) and also sets the background color of the box. Each of the nested span elements then applies another corner.

www.htmldog.com/examples/images3_2.html

More @media Master Classes
Griffiths: XHTML & CSS
Covering areas such as text, images, lists, forms, and table-free layout, and how to apply them following best-practices.
Keith: DOM Scripting
How to make exciting, functional enhancements

FIGURE 4.7 In this example, some extra HTML span tag "scaffolding" is necessary so that there is something to hook each corner onto.

More @media
Master Classes
Griffiths: XHTML & CSS
Covering areas such as text, images, lists, forms, and table-free layout, and how to apply them following best-practices.
Keith: DOM Scripting
How to make exciting, functional enhancements

FIGURE 4.8 With four separate corners, the box can accommodate different widths and heights…

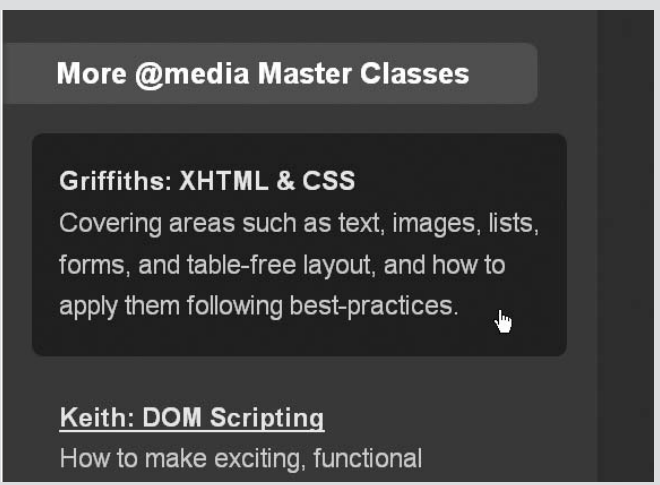

More @media Master Classes
Griffiths: XHTML & CSS
Covering areas such as text, images, lists, forms, and table-free layout, and how to apply them following best-practices.
Keith: DOM Scripting
How to make exciting, functional

FIGURE 4.9 …and if the user bumps up the text size, there isn't a problem.

Image Replacement: Providing Graphical Alternatives for Text

Image replacement is the process of using CSS to replace functional text with a graphical representation of that text. It has become an important part of web standards design, relegating `img` elements to a purely content-focused role in the same way that CSS layout has relegated tables.

A meaningful heading (for example) is simply something like "Plastic Banana Factory," which is easily sorted, as it should be, with text in HTML. If you want that heading presented with fancy yellow letters made up of bananas, for example, you shouldn't try and do that with HTML and an `img` tag because that carries no more meaning. What do we use for presentation, boys and girls? "CSS!" I hear you harmonize. Well done.

So the structured content is in place—simple, functional, accessible text in HTML. But we don't actually want to see that text—what we need to do is make it invisible and replace it with an alternative visual representation in the form of a CSS background image.

By keeping the images controlled by the CSS, you can also change them as you choose from one location. If you used it for a site-wide logo, for example, and the logo changes, you can swap the images globally with one small change to the CSS file. Rollovers too, where the image changes when the user moves the cursor over a link, can be achieved simply, without the need for JavaScript.

The CSS Zen Garden (*csszengarden.com*) is an excellent example of image replacement techniques, where the underlying HTML remains unchanged across all designs and includes no images at all. The headings are often replaced with images using CSS to achieve the desired look.

There are a number of ways to apply the technique. The basic idea is to hide the functional text somehow and then slap a background image in the "empty" box.

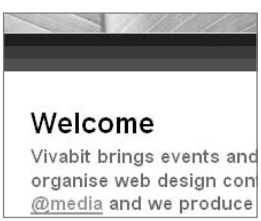


FIGURE 4.10 Before: "Welcome" as functional text (in a bold, Arial font)…

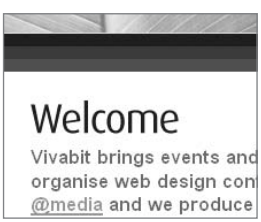


FIGURE 4.11 …and after: The "Welcome" text is pushed out of sight and replaced by a background image showing "Welcome" as graphical text, using the Dax font type.

We could start with HTML like this:

```
<h1><span>Sifaka</span></h1>
```

and then apply...

```
h1 {
    background-image: url(images/sifakalogo.gif);
    width: 300px;
    height: 129px;
}
h1 span {
    display: none;
}
```

The above example applies a background image to the `h1` element (which is made the same size as the image) and then the `span` element within it is hidden, hiding the text.

This traditional method is known as the Fahrner Image Replacement (FIR) technique. Unfortunately, it has one rather serious flaw. One of the supposed benefits of using image replacement techniques is that it aids accessibility. When `display: none` is used, however, not only can you not see that element, but most screen-readers will also ignore it; when a screen-reader comes across the FIR, it simply won't read anything at all.

The way around this isn't too difficult—you just need to use another way of hiding an element such as:

```
h1 span {
    display: block;
    height: 0;
    overflow: hidden;
}
```

(See Chapter 5 for more on these properties, in particular `display: none alternatives`.)

For all intents and purposes the `span` element within the `h1` is still displayed there—whereas to the eye zero height equals invisibility, to something that cares nothing for spatial parameters (such as a screen-reader) the element lives on in all its glory.

www.htmldog.com/examples/images4.html

Another image replacement method removes the need for the `span` tag scaffolding, so with the following slimline HTML:

```
<h1>The Sifaka</h1>
```

We can apply this CSS:

```
h1 {
    background-image: url(images/sifakalogo.gif);
    width: 300px;
    height: 129px;
    font-size: 1px;
    text-indent: -999em;
}
```

This applies the background image as before but by using a large negative `text-indent` the containing text is yanked out of view. The font-size is set to one pixel for the sole reason that otherwise it could push out the height of the `h1` element (any height less than the height of the image would do).

www.htmldog.com/examples/images5.html

The problem with these image replacement techniques is that they fail to show anything at all when images are turned off but CSS is on—the image won't load and the text will be hidden. The issue isn't only that it affects people who choose to switch off images for faster page downloads, but it also means that there is no placeholder text while the replacement image loads, so it doesn't have an advantage that the `img` alt text has.

Another image replacement technique gets around this CSS on/images off problem by reintroducing the `span` tag scaffolding, but in a slightly different arrangement:

```
<h1><span></span>Sifaka</h1>
```

To this we can apply the following CSS:

```
h1 {
    position: relative;
    width: 300px;
    height: 129px;
```

```
    font-size: 50px;
}
h1 span {
    position: absolute;
    top: 0;
    width: 300px;
    height: 129px;
    background-image: url(images/sifakalogo.gif);
}
```

This effectively lays the `span` element on top of the text in the `h1` element. The only restrictions are that the image background cannot be transparent (or else the underlying text will show through) and the text needs to be equal to or less than the size of the image (otherwise it will spill out from under the image).

`www.htmldog.com/examples/images6.html` demonstrates this method and shows up the problem of the necessity for a solid background: If the width of the browser is too narrow, the blue background of the logo will overlap the background of the page.

For a good rundown of the different techniques, hop on over to `mezzoblue.com/tests/revised-image-replacement/`

In theory, there are similar methods that can be used but do not require the `span` tag scaffolding, including manipulating the `:before` pseudo-element or, even easier, using the CSS3 `content` property (`h1 { content: url(images/sifakalogo.gif); }`, which replaces the content of an element, such as text, with something else, such as an image). Unfortunately, at the moment very few browsers (and Internet Explorer isn't one of them) can handle such Space Age methods, so there isn't much point in going into them here.

chapter 5

Layout

PRETTY TEXT and fancy images are all well and nice, but in terms of real layout—placing bits of a page exactly where you want them—things are a bit linear so far in this book.

Before CSS 2 became widely supported, the only practical way of laying out a page in anything other than a long single column was with HTML tables, transparent "spacer.gif" images, and lots of non-breaking spaces: nbsp;   …

Now that CSS 2 is widely supported, you can manipulate the position of every HTML element on a page with style sheets. Not only does this approach dramatically reduce page weight and download time (those multiple nested table elements and spacer images didn't half fatten things up), CSS layout also leads to more manageable, flexible, and uniform page layouts throughout a whole website from a single file. And, as a nice little bonus, it improves accessibility—thanks to the logical order of the underlying HTML (which isn't disturbed or compromised by presentational markup).

There's a fair bit to get through, but it'll all be worth it in the end. Starting with the basics of padding, borders, and margins of the box model through to the display property and positioning, this chapter ends with some practical examples to show how the theory can be brought together to achieve solid CSS page layouts.

The Box Model

Grand multicolumn page layouts might be your ultimate goal, but before moving on to see how that kind of thing can be achieved let's start with the basics of laying out elements: the box model. Every element on a web page is surrounded by a force field—a simple multi-layered box that can be manipulated to create sophisticated effects.

FIGURE 5.1 The mighty box model. At the center is the content itself. Surrounding that is the padding. Surrounding that is the border and surrounding that is the margin.

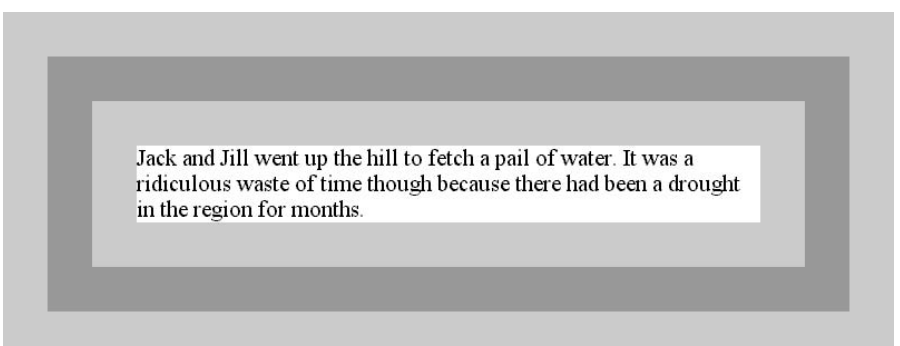

Jack and Jill went up the hill to fetch a pail of water. It was a ridiculous waste of time though because there had been a drought in the region for months.

FIGURE 5.2 The box model applies to all elements displayed on a web page. Paragraphs, for example…

FIGURE 5.3 …images…

FIGURE 5.4 …or lists, not to mention links, tables, forms, strong elements, etc., etc., etc.

Width and Height

You can set the width and height of an element using the `width` and `height` CSS properties.

These set the dimensions of the inner (content) box only, and do not take into account the padding, border, or margin. So if you set `width` to `100px` and have a 50-pixel border, 50-pixel padding, and 50-pixel margin, the total width of the box will actually be 400 pixels (100px + 50px left border + 50px right border + 50px left padding + 50px right padding + 50px left margin + 50px right margin).

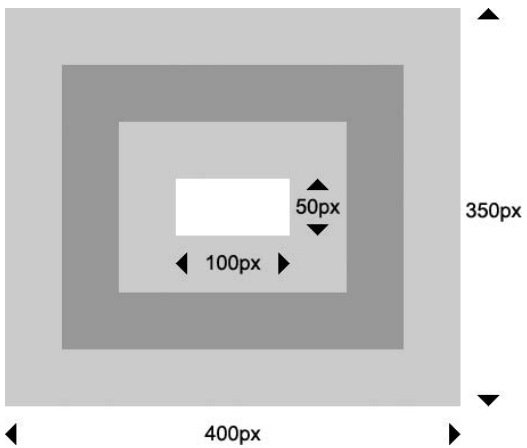


FIGURE 5.5 The dimensions of a box set to `width: 100px` (the width of the inner-most rectangle), `height: 50px` (the height of the innermost rectangle), `padding: 50px` (the rectangle around the innermost rectangle), `border: 50px solid` (the rectangle around that), and `margin: 50px` (the outermost rectangle).

The CSS standard also allows `min-width`, `min-height`, `max-width`, and `max-height` properties to set minimum and maximum widths and heights, but since Internet Explorer 6 doesn't support these properties, unfortunately they aren't a practical option at this time (although the problem is remedied in IE 7).

Overflow

If content is too large to fit into a box with a specified height and width, then the "overflow"—that portion of the content that doesn't fit in the box—can be set to do a number of things with the `overflow` property. This can be set to:

- `visible` (which is the default), whereby the overflow spills over the box.
- `hidden`, where any content that doesn't fit in the box will be "clipped"—cut off at the edge of the box.

- `scroll`, which displays scrollbars, allowing the user to scroll the box to see the overflow.
- `auto`, which displays scrollbars only if they are necessary (whereas `overflow: scroll` will show them even if the content of the box fits without any overflow).

Internet Explorer slips into a state of undignified discombobulation when it comes to `overflow:visible` (which is the default behavior on all boxes). It will expand the box's height beyond that specified, effectively interpreting height as min-height should be interpreted.

The `clip` property can also be used with absolutely positioned boxes (described later in this chapter) to specify an area of a box that is visible.

`clip: rect(10px 120px 120px 10px)`, for example, will clip a region starting at 10px in from the left and 10px in from the top and end at 120px in from the left (the "right" part) and 120px from the top (the "bottom" part), leaving a 110px by 110px area.

Padding

Individual sides of the box can be padded by using the `padding-top`, `padding-right`, `padding-bottom`, and `padding-left` properties (for example, `padding-top: 2em`).

If you want to set the padding on more than one side, though, you can use the `padding` property, with which you can apply different amounts of padding to each side of a box if you so choose (see "Shorthand Values" sidebar).

```
#header {
    padding-bottom: 1em;
}
#content {
    padding: 1em 2em;
}
```

Backgrounds, be they images or colors, will fill the area of the content and the padding.

SHORTHAND VALUES

With `padding`, `border-width`, and `margin` you can provide a single value to specify uniform padding/border width/margin in a box. To set different values for different sides you could use properties like `padding-top` and `padding-left` or `border-right-width` and `border-bottom-width`, for example, but you can also apply different values to different sides of a box in one shorthand property. By specifying two, three, or four values to padding, border-width, or margin, you can target different sides as the following table indicates:

Values	Example	Applies to
1	`padding: 1em`	all sides
2	`margin: 10px 2em`	[top and bottom] [left and right]
3	`border-width: 1px 5px 2px`	[top] [right and left] [bottom]
4	`padding: 10px 10px 1em 1em`	[top] [right] [bottom] [left](clockwise)

Note that the values don't all have to be the same—you can mix up pixels, ems, and more if that floats your boat.

Borders

Borders have a bit more to them than padding because not only can you specify their width, you can specify their style and color.

Border width works much in the same way padding does—you can specify measurements for individual sides (using `border-top-width`, `border-right-width`, `border-bottom-width`, and `border-left-width`) or you can specify multiple sides at once using border-width with one, two, three, or four values.

Before `border-width` does anything, however, you need to specify what kind of border you want with `border-style`.

Some values for `border-style`, which are as mad as a particularly mad clown in a mental asylum, include `groove`, `ridge`, `inset`, and `outset`. These render differently

in different browsers, look pretty nasty due to their generic "embossed" style anyway, and, therefore, are almost as useful as a saucepan made out of cream cheese.

This property's most commonly used values are the self-descriptive `solid`, `dotted`, or `dashed`, which are a tad more useful. Whereas browsers will normally render dotted borders as a series of equally separated dots, Internet Explorer 6 (and earlier versions), in an interesting quirk, will render them as dashed lines if the border is 1 pixel wide.

You can specify different styles for different sides of the border using `border-top-style`, `border-right-style`, `border-bottom-style`, and `border-left-style` or specify more than one value with `border-style` (following the same principles as the `border-width` shorthand—see sidebar). `border-style: solid dotted dashed solid`, for example, will apply a solid border to the top, a dotted border to the right, a dashed border to the bottom and a solid border to the left of the box.

`border-color` (and `border-top-color`, `border-right-color`, `border-bottom-color`, and `border-left-color`) can be used to change the color of a border.

If a border color is not specified, it will assume the color of the `color` property of the box.

You can further simplify things by combining border settings with the `border` shorthand property, which allows you to set the `border-width`, `border-style`, and `border-color` styles in one handy property. `border-top`, `border-right`, `border-bottom`, and `border-left` achieve the same things for individual sides of the border.

`border: 1px solid black`, for example, will set a one-pixel-wide solid black border to a box. If you don't want a uniform border, you can follow the border declaration with separate specific declarations:

```
#orangutan {
    border: dotted red;
    border-width: 2px 10px;
}
#chimpanzee {
    border: 2px solid;
    border-color: black #333 #666 #999;
}
```

www.htmldog.com/examples/border.html

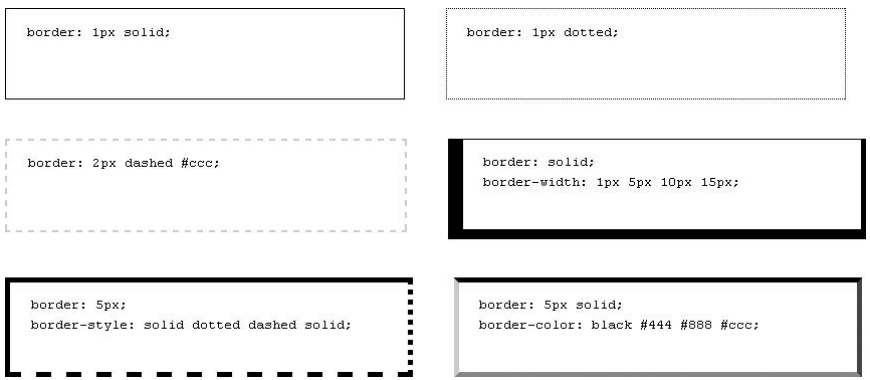


FIGURE 5.6 Some examples of different border styles and property combinations. See *www.htmldog.com/examples/borders.html.*

Margin

And so to margins—the transparent outer wrapping of the box. We return to the simplicity of padding in terms of defining them—you simply specify the width. Once more, you can set the sides individually with `margin-top`, `margin-right`, `margin-bottom`, and `margin-left`, or use `margin` with one, two, three, or four values:

```
#bonobo {
    margin-top: 1em;
}
#human {
    margin: 3em 1em;
}
```

Margin Collapsing

If two or more vertical margins come into contact, a phenomenon known as margin collapsing will occur. The distance between two boxes will be the distance of the greater of the two margins, rather than the sum of both. The smaller margin will "collapse" and disappear—only the larger margin will remain to space out the two boxes.

So if you had a number of paragraphs, one after the other, and their margin was set to 1em, the margin between each paragraph would be 1em, not 2.

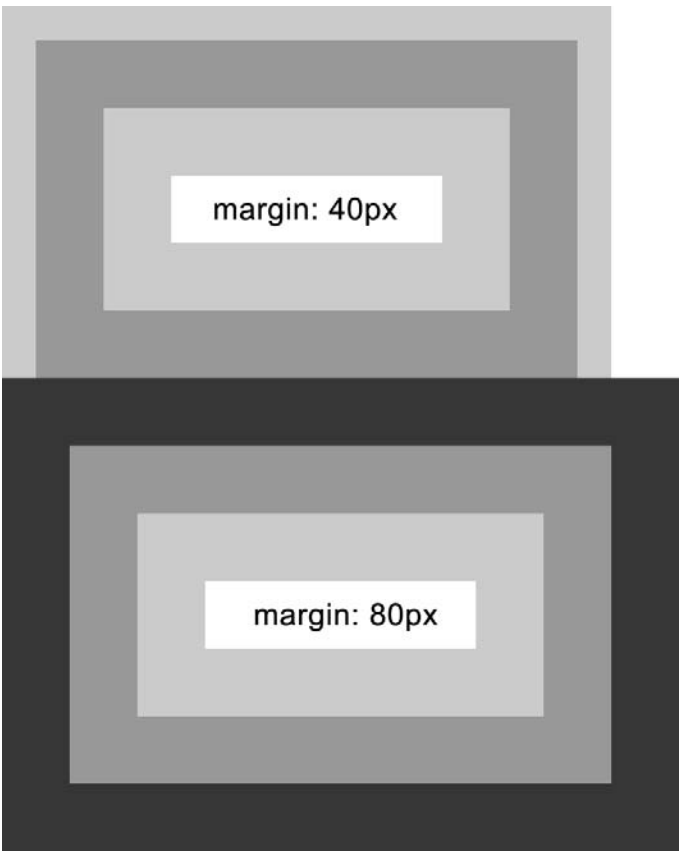


FIGURE 5.7 "Margin collapsing": When two vertical margins come into contact, only one will apply.

Margin collapsing happens not only to boxes that follow each other, but also within boxes. So if you were to have a paragraph with a 1em margin inside a `div` with a 1em margin, the paragraph margins would disappear into the margin of the `div`.

This element-within-element collapsing only occurs when there is no other box level between the margins. Using the last example, if the `div` has any padding or border applied to it, then that will act as a boundary, and the paragraph margins will not come into contact with the div margins, preventing them from collapsing.

www.htmldog.com/examples/margincollapse.html

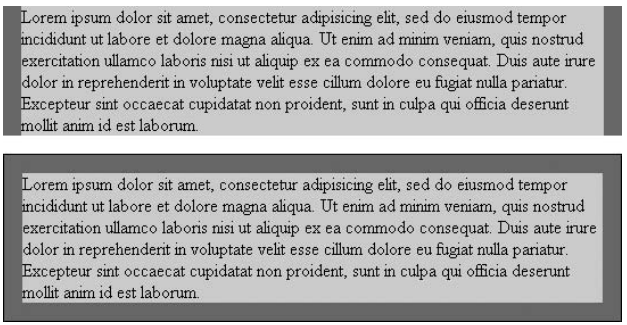

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

FIGURE 5.8 Both paragraphs and divs (dark background) have a margin of 1em. The div in the second block has a 1px border, preventing the margin collapse with the paragraphs.

THE BOX MODEL HACK

Internet Explorer 5.0 and 5.5 for Windows handle the box model incorrectly.

Instead of applying `width` and `height` properties to the inner content box, they will be applied to the content box plus the padding plus the border.

FIGURE 5.9 How IE 5.x renders a box set to `width: 200px; padding: 20px; border: 3px solid;`.

So when you are applying padding and borders to an element, you need to specify a different height and width for IE 5.x than you do for other browsers. There is no "proper" way of doing this because CSS (and HTML, for that matter) is supposed to be browser-independent. In the real world, however, practical problems like this sometimes arise, and that's when we resort to a hack to tackle

them. The Box Model Hack can take a number of forms, but the simplest goes something like this:

```
#somebox {
     width: 200px
     wid\th: 154px;
     padding: 20px;
     border: 3px solid;
}
```

Basically, IE 5.x won't recognize property names that are "escaped" in this way in the middle. In this case it won't recognize "wid\th" as "width," whereas other browsers will.

So by specifying the incorrect width first with width (that all browsers, including IE 5.x, will understand) and then the correct width afterwards with wid\th (that all browsers except IE 5.x will understand), IE 5.x will apply the first "incorrect" width declaration (because it won't understand the second) and all other well-behaved browsers will apply the "correct" width (because they will understand both declarations and give preference to the latest one).

In fact, you can use this hack with any property. As long as the "\" doesn't come before an a, b, c, d, e, or f (which conflicts with the hex codes used in colors), it will work.

There are a number of hacks that will hide various things from various browsers (*dithered.com/css_filters* is one source for a good, comprehensive outline) but they should be avoided if at all possible. There is rarely a need to use them—one of the beauties of web standards is that by using them you can feel safer in cross-browser compatibility and reliability. The Box Model Hack, however, is the most important and most widely used of hacks because it deals with such a big fault in a commonly used technique in a commonly used browser.

The `display` Property

Boxes can be block or inline. These terms are derived from block and inline elements (see Chapter 1, "Getting Started") whereby a block element (such as `p` or `div`) is displayed with a line-break before and after it (a block box) by default and an inline element (such as `em` or `span`) is displayed on the same line (an inline box) by default. But these presentational aspects need not apply to specific elements—you can take any element and display it any way you choose—either in a block box or an inline box.

Block boxes, as well as starting on a new line and forcing anything following them to start on a new line, will also stretch to fit the width of their containing box (unless an explicit width is specified). The width of an inline box equates to the width of the content.

www.htmldog.com/examples/blockinline1.html

Control over the box type is an important aspect of gaining complete visual control over your pages. You don't want to compromise the markup, because you should apply meaningful tags where appropriate, but you might not want to accept the default rendering of the element. You might choose to have the links in a navigation bar displayed as block boxes or headings displayed inline, for example. Another example is one of the methods for creating horizontal lists, as described in Chapter 6, "Lists," where list items have their display style changed so that they are side-by-side instead of on different lines.

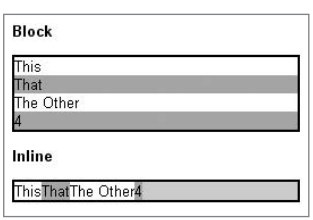


FIGURE 5.10 Block boxes will start on new lines and stretch to fit the width of the containing box, whereas inline boxes will remain on the same line and only be as wide as the content (see *www.htmldog.com /examples/blockinline1.html*).

Additionally, inline boxes handle vertical parameters differently than block boxes. Vertical margins are not applied and padding and borders, rather than pushing out content above or below, will spill over the line and lay on top of anything above and

below it. Note, however, that IE 5.0 does not apply padding, borders, or margins to inline boxes at all.

www.htmldog.com/examples/blockinline2.html

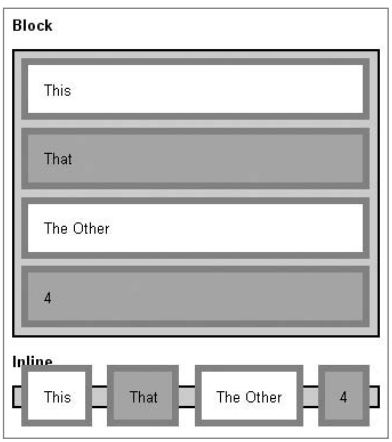


FIGURE 5.11 With padding, border, and margin applied, block boxes will behave as you might expect (also note the margin collapsing). Inline boxes, however, will ignore vertical margins and the padding and border will spill over the line (see *www.htmldog.com/examples/blockinline2.html*).

You can set any element you want to be any type of box you choose using the `display` property.

The most common values are `block` and `inline`, for your basic block and inline boxes, but you can also use `none` (which doesn't render the box at all, basically pulling it out from the page).

The other, much less used (and supported), display types are variations on the block and inline theme. `list-item` and the various table-related components such as `table` and `table-row` can perhaps best be understood by looking at a browser's default rendering of the equivalent HTML elements (such as `li`, `table`, and `tr` elements—see the `display` property in the CSS Appendix for more).

`inline-block` works like a block box wrapped in an inline box. These boxes have the same characteristics as an inline box (staying on the same line and being the width of the content) but the vertical padding, border, and margin work the same way as a block box, pushing out that which surrounds it. Unfortunately, this display type is not supported by Mozilla and is somewhat buggy in Internet Explorer (IE will only apply it if the element in question is an inline element).

www.htmldog.com/examples/blockinline3.html

THE PROBLEM WITH `display: none`

Sometimes you'll have something in your HTML that you might not want in the visual display of the page. It is sometimes an accessibility consideration (such as the "skip navigation" link mentioned in Chapter 3, "Links") or it could be that you want to maintain a flexible document, leaving open different styling options (such as not wanting certain elements displayed when printed—see Chapter 10, "Multiple Media"). Both of these boil down to the idea that the HTML should work as a structured, CSS-independent document with all of the meaningful pieces in place, regardless if you want them seen or not.

`display: none` would seem an obvious candidate to remove elements from sight—it does exactly what it says. The only trouble is that it does a bit more too. Not only does `display: none` hide an element from sight, it also hides it from screen-readers. It won't be seen, but it won't be heard, either. So if you're hiding some information that should actually remain accessible to screen-readers, this isn't much help.

What about `visibility: hidden`? It makes an element invisible (rather than removing it completely as `display: none` does), but it has the same problem as `display: none`, rendering the element invisible to screen-readers as well. It also leaves behind a space where the element would normally be seen, which usually isn't desirable.

Since these obvious options are out, we need to try something less obvious. The solution is to keep the element *hypothetically* visible, but with zero width and/or height:

```
.accessaid {
    position: absolute;
    height: 0;
    overflow: hidden;
}
```

There are variations on this CSS, but the principle is the same. To the eye, zero height or zero width equals invisibility, but to something that cares nothing for spatial parameters (such as a screen-reader) the element lives on in all its glory.

When the element box itself is needed for CSS (such as with image replacement techniques—See Chapter 4, "Images"), the content alone can be smacked out of view with a negative `text-indent`:

```
h1 {
    text-indent: -999em;
}
```

Positioning

Boxes can be positioned on a page in various ways—statically, relatively, absolutely, or fixed using the `position` property.

Static

By default, element boxes are static. Static boxes follow the normal flow of the page, immediately following and preceding other static elements (the boxes in Figures 5.10 and 5.11 are all static, for example).

www.htmldog.com/examples/positioning1.html

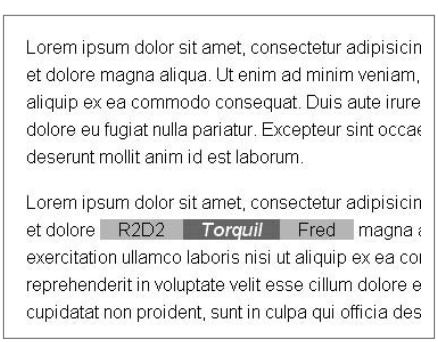

Lorem ipsum dolor sit amet, consectetur adipisicin
et dolore magna aliqua. Ut enim ad minim veniam,
aliquip ex ea commodo consequat. Duis aute irure
dolore eu fugiat nulla pariatur. Excepteur sint occa
deserunt mollit anim id est laborum.
Lorem ipsum dolor sit amet, consectetur adipisicin
et dolore R2D2 Torquil Fred magna
exercitation ullamco laboris nisi ut aliquip ex ea co
reprehenderit in voluptate velit esse cillum dolore e
cupidatat non proident, sunt in culpa qui officia des

FIGURE 5.12 A static box inside a static paragraph.

Relative

A relatively positioned box is one that can be moved to a position that is relative to its initial position, leaving an empty space where it once was. When a box is made relative, you can specify values for `top`, `right`, `bottom`, or `left`, from which the box will be offset. The values tell a browser how far to offset the box from that position, so `position: relative; left: 10em`, for example, will push the box 10 ems to the right (10 ems from the left of the initial position). By the same token, `position: relative; right: 10em` will push the box 10 ems to the left.

The position of any boxes that follow a relatively positioned box will be calculated from the initial position of the offset box, not from the offset position. So the position of a box (let's call it Fred) that follows a relative box (which we'll call Torquil) will not be calculated from the actual position of Torquil's box, but rather from the original position, as if Torquil wasn't offset.

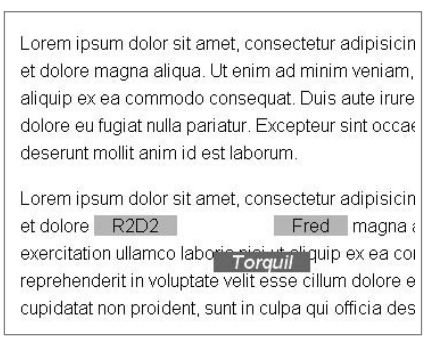

Lorem ipsum dolor sit amet, consectetur adipisicin
et dolore magna aliqua. Ut enim ad minim veniam,
aliquip ex ea commodo consequat. Duis aute irure
dolore eu fugiat nulla pariatur. Excepteur sint occae
deserunt mollit anim id est laborum.
Lorem ipsum dolor sit amet, consectetur adipisicin
et dolore R2D2 Fred magna
exercitation ullamco
Torquil
reprehenderit in voluptate velit esse cillum dolore e
cupidatat non proident, sunt in culpa qui officia des

FIGURE 5.13 A relatively positioned box (`position: relative; top: 2em; left: 2em;`) inside a paragraph.

www.htmldog.com/examples/positioning2.html

Absolute

Boxes can also be positioned absolutely. Unlike a relative box, an absolute box is offset from the position of its containing block, which will be the page itself unless the box exists inside a relative or absolute block, in which case it will be offset from that.

`position: absolute` makes a box absolute and, once more, `top`, `right`, `bottom`, and `left` are used to position it.

www.htmldog.com/examples/positioning3.html

Absolute boxes are pulled out of the normal flow of a page, existing independently from the rest of the content. Whereas relative boxes leave behind the space where the box once was, the position of a box that follows an absolute box (which we'll again call Torquil—he did such a great job) will be calculated from the start of Torquil's original position, as if Torquil didn't even exist.

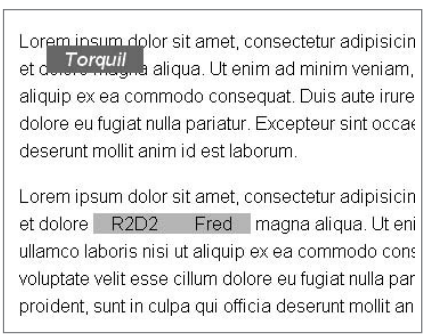

Torquil
aliqua. Ut enim ad minim veniam,
aliquip ex ea commodo consequat. Duis aute irure
dolore eu fugiat nulla pariatur. Excepteur sint occae
deserunt mollit anim id est laborum.
et dolore R2D2 Fred magna aliqua. Ut eni
ullamco laboris nisi ut aliquip ex ea commodo cons
voluptate velit esse cillum dolore eu fugiat nulla par
proident, sunt in culpa qui officia deserunt mollit an

FIGURE 5.14 An absolutely positioned box (`position: absolute; top: 2em; left: 2em;`) inside the same static paragraph.

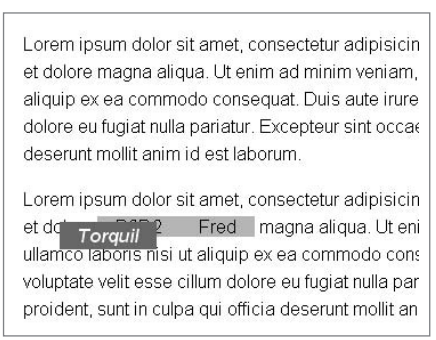

et dolore magna aliqua. Ut enim ad minim veniam,
aliquip ex ea commodo consequat. Duis aute irure
dolore eu fugiat nulla pariatur. Excepteur sint occae
deserunt mollit anim id est laborum.
Fred magna aliqua. Ut eni
Torquil
ullamco laboris nisi ut aliquip ex ea commodo cons
voluptate velit esse cillum dolore eu fugiat nulla par
proident, sunt in culpa qui officia deserunt mollit an

FIGURE 5.15 The same absolutely positioned box inside a relatively positioned paragraph.

www.htmldog.com/examples/positioning4.html

Fixed

Fixed boxes (`position: fixed`) are similar to absolutely positioned boxes, apart from the fact that they are fixed to the viewport. Like background images set to `background-attachment: fixed` (see Chapter 4), fixed boxes will not scroll when the rest of the content does. Unlike fixed backgrounds, though, fixed boxes are not supported by Internet Explorer 6 (though they *are* supported in IE 7).

THE Z-INDEX

Because positioned boxes are pulled out of the normal flow and can sit on top of one another, you may want to control which of these boxes appears where in this stacking order. Suddenly we have three dimensions to think about—we have the x-axis that governs where something is horizontally, the y-axis where something is vertically, and now we have the z-axis, which governs depth. The x and y axes are controlled by `width`, `height`, `left`, `right`, `top`, `bottom`, `padding`, `margin`, and so on, but we don't need anything so elaborate with the z-axis, we just need to state the order in which things appear on top of each other.

Like Mighty Mouse, `z-index` is here to save the day.

This property is used to specify where in the stacking order a positioned box should be. The higher the number, the higher the box is in the stack. `z-index: 3` will be below `z-index: 5` but above `z-index: 1`, for example.

Floating

Floating, using the `float` property, is another method that can be used to push around boxes and manipulate how others respond to them.

A floated box will basically push the box to the far left (`float: left`) or right (`float: right`) of its container and cause surrounding content to flow around it

rather than continue underneath it. A floated box will override any `display` type setting and render the box as a block box.

www.htmldog.com/examples/float1.html

www.htmldog.com/examples/float2.html

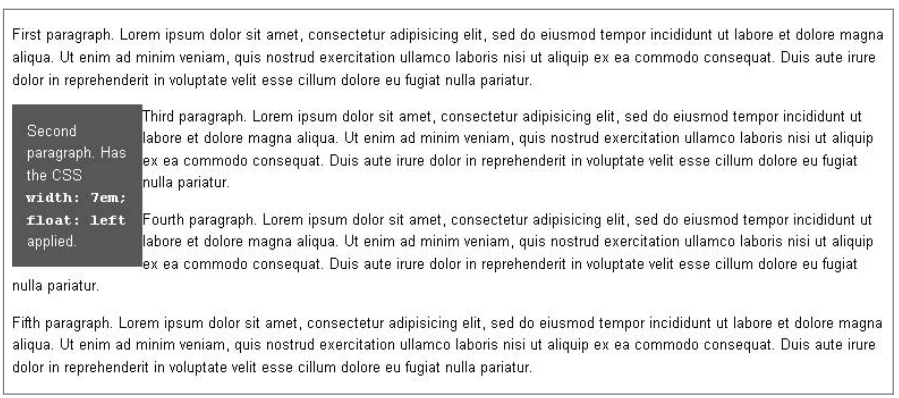

First paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Second paragraph. Has the CSS width: 7em; float: left applied.
Third paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Fourth paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Fifth paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

FIGURE 5.16 A left-floated paragraph. See *www.htmldog.com/examples/float1.html*.

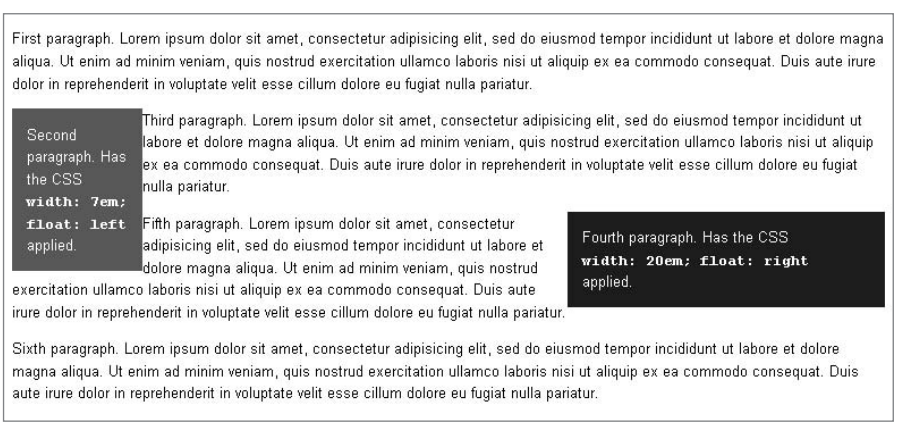

First paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Second paragraph. Has the CSS width: 7em; float: left applied.
Third paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Fifth paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Fourth paragraph. Has the CSS width: 20em; float: right applied.
Sixth paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

FIGURE 5.17 A left-floated paragraph and a right-floated paragraph. See *www.htmldog.com/examples/float2.html.*

If you want an element that follows a floating box to start underneath the floated box, rather than flow around it, you can use the property `clear`.

`clear: left` will clear all left-floated boxes, `clear: right` will clear all right-floated boxes, and `clear: both` will do something I'm sure you'd never expect.

www.htmldog.com/examples/float3.html

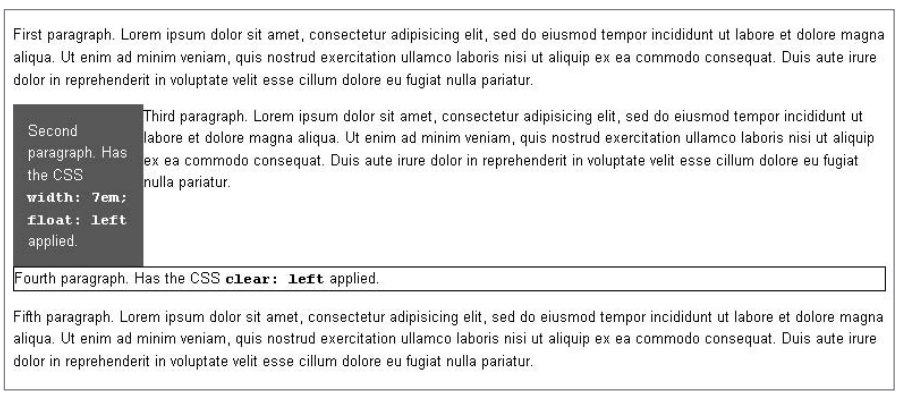

First paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Second paragraph. Has the CSS width: 7em; float: left applied.
Third paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Fourth paragraph. Has the CSS clear: left applied.
Fifth paragraph. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

FIGURE 5.18 The fourth paragraph is set to `clear: left` and so starts underneath the left-floated paragraph rather than flowing around it. See *www.htmldog.com/examples/float3.html.*

Essentially, clearing works by increasing the top margin of the cleared box enough so that it will start below the floated element. Because of this, the rules of margin collapsing should be remembered: If the cleared box has a top margin explicitly applied, it will only work if that margin is larger than the height of the floated box. Then the margin will apply from the position of the box before it was cleared, rather than from the bottom of the floated box.

For a few simple techniques involving floats, take a gander at *www.htmldog.com/articles/dropcaps/* and *www.htmldog.com/articles/pullquotes/*, which are accompanied by a few bare-bone examples.

ABSOLUTE VS. RELATIVE VALUES II

In Chapter 2, "Text," the pros and cons of using absolute units such as pixels and relative units such as ems are looked at in relation to text. But the choice is there for the dimensions of all boxes on a web page, and the choice between absolute and relative values can lead to vastly different results.

There are three approaches to defining the sizes of boxes—fixed, liquid, and elastic.

Fixed Layout

In fixed layouts, the widths of areas of the layout are explicitly specified in lengths (rather than not specified at all or specified in percentages), which are usually defined using pixels as units.

The advantage of fixed layouts is that the width of lines of text can be constrained to keep them easy to read—the line length will remain unchanged no matter what size the user's window. The relationship between text area widths and image widths can also be maintained (if you have a 500px-wide image, for example, then a 500px-wide paragraph can be used to complement it).

The main disadvantage of fixed layouts is that they don't take advantage of the full area of the screen, leaving wasted space and a greater likelihood that a user will have to scroll to reach more content.

FIGURE 5.19 Jon Hicks' blog (*hicksdesign.co.uk/journal*) utilizes a 900px fixed-width design.

FIGURE 5.20 In a wider window the content area remains the same width even if the text-size changes.

Liquid Layout

In liquid, or fluid, layouts, the widths of some or all areas of the layout are specified in percentages or not specified at all, so that the layout will stretch or shrink, depending on the size of the browser window.

The advantage of liquid layouts is that they take full advantage of a user's computer display capabilities. Users who have large monitors can stretch their browser windows to show more content "above the fold." The flexible nature means that not only do fluid layouts take advantage of larger screens, they also have more chance of working on smaller screens, such as those on PDAs or the latest generation of mobile phones (see Chapter 10, "Multiple Media").

The disadvantage of a liquid layout is that longer lines, which can come about on larger displays, can be more uncomfortable to read. From a graphic design point of view, size-based relationships with fixed-width objects, such as images, are also difficult.

FIGURE 5.21 Dan Webb's site (*danwebb.net*) employs a content area that has a liquid width.

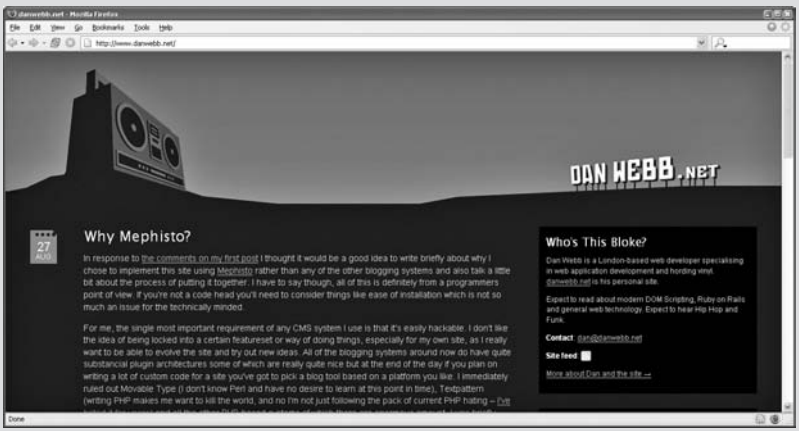


FIGURE 5.22 The content area will stretch to fill the width of the window.

Elastic Layout

Elastic layout is a cousin of fixed and liquid layouts. It involves using relative units such as ems, rather than absolute values such as pixels, to define widths, so that the entire layout will expand and contract depending on the text-size preferences of the user's browser. This approach can use elements from fixed and fluid layouts, whereby areas can either be fixed (where all widths are defined in ems) or fluid (where only some widths, such as those of navigation columns, are defined in ems).

Elastic layouts can aid accessibility by increasing proportions as well as text size, making areas of text more comfortable to read for those who are visually impaired and need to bump up the text size. By maintaining text-size to box-dimension ratios constant, this approach can also prevent things such as unwanted line wrapping, due to containing boxes increasing in size with their contents.

The downside is that a fixed-elastic layout can become too wide if the text size is increased too much. A good rule of thumb is to check that it will display on an 800px-wide screen on Internet Explorer's "largest" text-size setting.

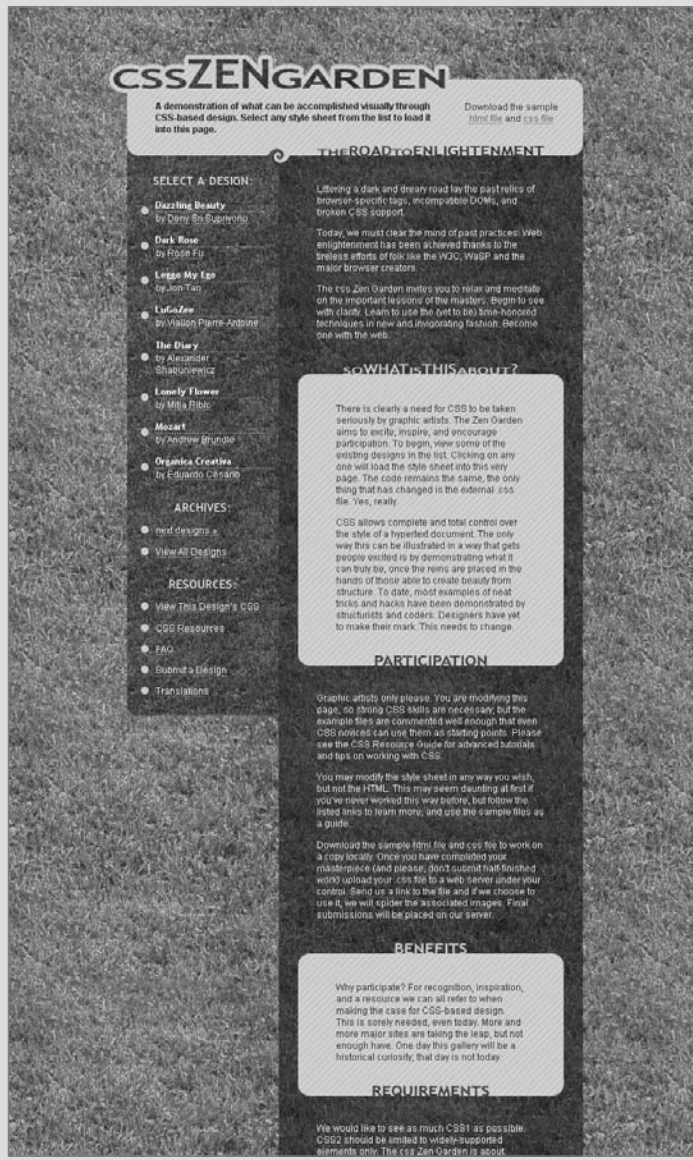

CSSZENGARDEN
A demonstration of what can be accomplished visually through CSS-based design. Select any style sheet from the list to load it into this page.
Download the sample html file and css file
THE ROAD TO ENLIGHTENMENT
SELECT A DESIGN:
Dazzling Beauty by Dany Sri Supriyono
Dark Rose by Rose Fu
Lego My Ego by Jon Tan
LuGoZee by Viallon Pierre-Antoine
The Diary by Alexander Shaburnevicz
Lonely Flower by Mitja Ribic
Mozart by Andrew Brundle
Organica Creativa by Eduardo Cesario
ARCHIVES:
next designs »
View All Designs
RESOURCES:
View This Design's CSS
CSS Resources
FAQ
Submit a Design
Translations
Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.
Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.
The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.
SO WHAT IS THIS ABOUT?
There is clearly a need for CSS to be taken seriously by graphic artists. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The code remains the same, the only thing that has changed is the external .css file. Yes, really.
CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way that gets people excited is by demonstrating what it can truly be, once the reins are placed in the hands of those able to create beauty from structure. To date, most examples of neat tricks and hacks have been demonstrated by structurists and coders. Designers have yet to make their mark. This needs to change.
PARTICIPATION
Graphic artists only please. You are modifying this page, so strong CSS skills are necessary, but the example files are commented well enough that even CSS novices can use them as starting points. Please see the CSS Resource Guide for advanced tutorials and tips on working with CSS.
You may modify the style sheet in any way you wish, but not the HTML. This may seem daunting at first if you've never worked this way before, but follow the listed links to learn more, and use the sample files as a guide.
Download the sample html file and css file to work on a copy locally. Once you have completed your masterpiece (and please, don't submit half-finished work) upload your .css file to a web server under your control. Send us a link to the file and if we choose to use it, we will spider the associated images. Final submissions will be placed on our server.
BENEFITS
Why participate? For recognition, inspiration, and a resource we can all refer to when making the case for CSS-based design. This is sorely needed, even today. More and more major sites are taking the leap, but not enough have. One day this gallery will be a historical curiosity, that day is not today.
REQUIREMENTS
We would like to see as much CSS1 as possible. CSS2 should be limited to widely-supported elements only. The css Zen Garden is accut

FIGURE 5.23 The "Elastic Lawn" design on the CSS Zen Garden (see *csszengarden.com/?cssfile=063/063.css*) at the "normal" text-size setting.

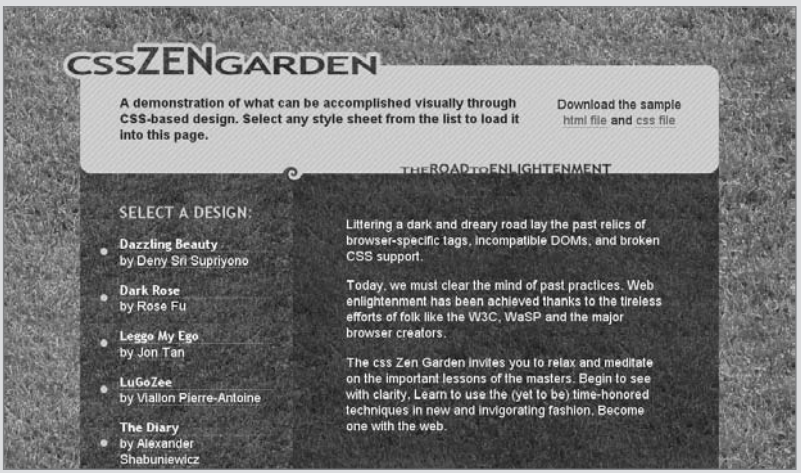

CSSZENGARDEN
A demonstration of what can be accomplished visually through CSS-based design. Select any style sheet from the list to load it into this page.
Download the sample html file and css file
THE ROAD TO ENLIGHTENMENT
SELECT A DESIGN:
Dazzling Beauty by Dany Sri Supriyono
Dark Rose by Rose Fu
Leggo My Ego by Jon Tan
LuGoZee by Viallon Pierre-Antoine
The Diary by Alexander Shabuniewicz
Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.
Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.
The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.

FIGURE 5.24 When the text size is changed, the dimensions of the layout will change as well.

You can read more about elastic layouts at *www.htmldog.com/articles/elasticdesign/.*

"Which layout method is better?" is one of the biggest arguments in web design. When it comes down to it, though, there is no one correct answer. Each of these approaches has its advantages and disadvantages and whereas one may be more appropriate for one website, a different approach may be more appropriate for another site.

Sample Page Layouts

Swell. Now we've got the theory sussed, let's put it to practice on a grand scale: page layouts. All it takes is a combination of manipulating the box model, positioning, and floating.

The thought process behind laying out a page should go a little something like this: "Right. I've got this chunk of content and I want it there. So I'll just shove that there. And I've got this chunk of content and I want that over there. Cool. I'll just shove that there and shift it along a bit. Excellent." CSS layout is all about grabbing chunks of HTML and placing them wherever you want on the page.

The initial examples that we'll go through here—creating columns, headers, and footers—are simple bare-bones layouts used to demonstrate the techniques without any extra bells and whistles getting in the way and confusing things. With these basic arrangements you can flesh things out and make your pages look as appealing as your imagination will allow.

And although these examples are used in the context of page layouts, the same techniques can be used with any part of a page—remember, we're just talking about chunks here. It doesn't matter if they're large or small.

Creating Columns

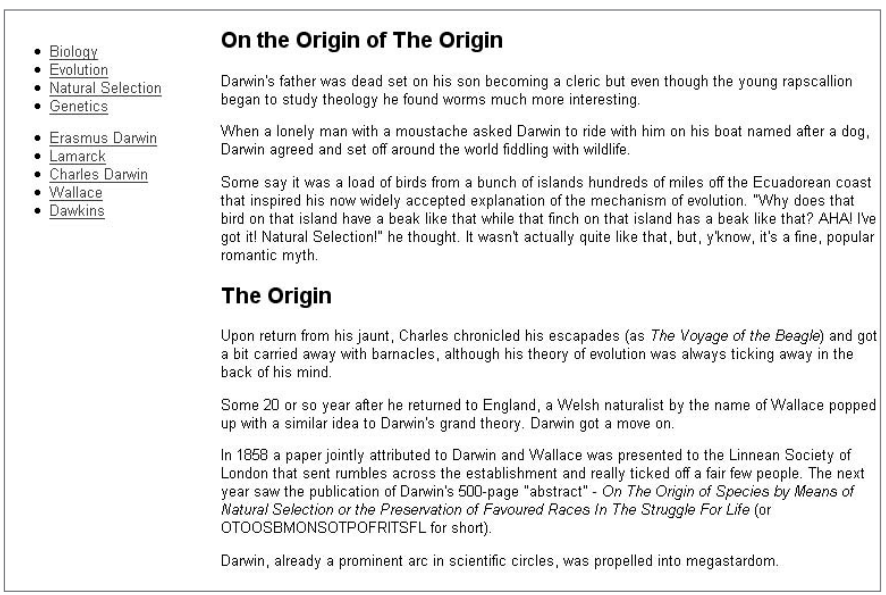

Biology
Evolution
Natural Selection
Genetics
Erasmus Darwin
Lamarck
Charles Darwin
Wallace
Dawkins
On the Origin of The Origin
Darwin's father was dead set on his son becoming a cleric but even though the young rapscallion began to study theology he found worms much more interesting.
When a lonely man with a moustache asked Darwin to ride with him on his boat named after a dog, Darwin agreed and set off around the world fiddling with wildlife.
Some say it was a load of birds from a bunch of islands hundreds of miles off the Ecuadorean coast that inspired his now widely accepted explanation of the mechanism of evolution. "Why does that bird on that island have a beak like that while that finch on that island has a beak like that? AHA! I've got it! Natural Selection!" he thought. It wasn't actually quite like that, but, y'know, it's a fine, popular romantic myth.
The Origin
Upon return from his jaunt, Charles chronicled his escapades (as The Voyage of the Beagle) and got a bit carried away with barnacles, although his theory of evolution was always ticking away in the back of his mind.
Some 20 or so year after he returned to England, a Welsh naturalist by the name of Wallace popped up with a similar idea to Darwin's grand theory. Darwin got a move on.
In 1858 a paper jointly attributed to Darwin and Wallace was presented to the Linnean Society of London that sent rumbles across the establishment and really ticked off a fair few people. The next year saw the publication of Darwin's 500-page "abstract" - On The Origin of Species by Means of Natural Selection or the Preservation of Favoured Races In The Struggle For Life (or OTOOSBMONSOTPOFRITSFL for short).
Darwin, already a prominent arc in scientific circles, was propelled into megastardom.

FIGURE 5.25 A basic two-column page.

Two columns are all we need to tie down the basics of page layout. By following these simple principles, it isn't difficult to take the next step and move all manner of chunks around a page.

First we need to start with a well-structured document. There is a tag specifically designed to divide up large chunks of content—`div` (see Chapter 1), and this is

usually the best choice for defining important chunks. We can then latch on to their IDs with CSS and move them about.

```
<div id="navigation">
    <!--stuff-->
</div>
<div id="content">
    <!--stuff-->
</div>
```

For this example, we want the navigation to be a thin column in comparison to the main content, so we need to explicitly set the width of it. Then, to move it to the left we need to yank it out of the flow using absolute positioning:

```
#navigation {
    position: absolute;
    left: 0;
    width: 30em;
}
```

FIGURE 5.26 The navigation box now sits on top of the content.

www.htmldog.com/examples/pagelayout1.html

Things are starting to take form (no, honestly, they are), but at the moment the navigation is sitting on top of the content, which isn't all too helpful. So how can we shift all of the content into view? Well, there's a number of ways, but the most

obvious is to simply set the left margin of the content div to push the content past the width of the navigation area:

```
#content {
    margin-left: 30em;
}
```

The result? Two separate areas, just like Figure 5.25.

www.htmldog.com/examples/pagelayout2.html

A Solid Navigation Column

To visually separate the two parts of a page, you might want to specify a background color for the navigation column, but sometimes the content area will be much longer and the background color will of course end at the bottom of the navigation box (as in Figure 5.25). Ideally, you would want the bottom of the navigation area to line up with the bottom of the content area.

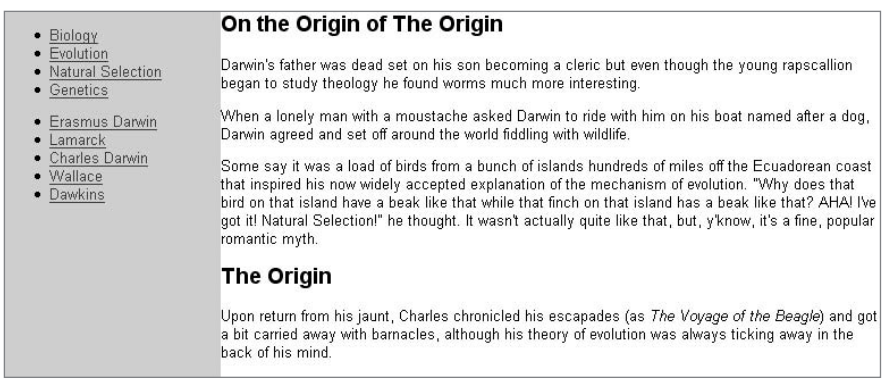

Biology
Evolution
Natural Selection
Genetics
Erasmus Darwin
Lamarck
Charles Darwin
Wallace
Dawkins
On the Origin of The Origin
Darwin's father was dead set on his son becoming a cleric but even though the young rapscallion began to study theology he found worms much more interesting.
When a lonely man with a moustache asked Darwin to ride with him on his boat named after a dog, Darwin agreed and set off around the world fiddling with wildlife.
Some say it was a load of birds from a bunch of islands hundreds of miles off the Ecuadorean coast that inspired his now widely accepted explanation of the mechanism of evolution. "Why does that bird on that island have a beak like that while that finch on that island has a beak like that? AHA! I've got it! Natural Selection!" he thought. It wasn't actually quite like that, but, y'know, it's a fine, popular romantic myth.
The Origin
Upon return from his jaunt, Charles chronicled his escapades (as The Voyage of the Beagle) and got a bit carried away with barnacles, although his theory of evolution was always ticking away in the back of his mind.

FIGURE 5.27 Using a colored border to shift along the content area and serve as a background for the navigation column.

`margin` seems like the obvious choice to shunt across the content `div`, as we have used above, but you can use any part of the box model—you also have `padding` and `border` at your disposal. By using `border` you can also color that area, which

will give the impression of a solid navigation bar that runs as long as the content area does.

```
#navigation {
    position: absolute;
    left: 0;
    width: 30em;
    background: #ccf;
}
#content {
    border-left: 30em #ccf solid;
}
```

www.htmldog.com/examples/pagelayout3.html

Another way to get around the short navigation bar problem is to set the background color of the containing block (such as the page) to the color you want for the navigation area (yellow, for example) and then color the rest of the boxes (such as the header and the content) to the main color you want for the page (such as white). You can use the same technique with background images, too.

Floating Column

Alternatively to positioning, you can also float your page chunks.

```
#navigation {
    left: 0;
    width: 30em;
}
#content {
    float: right;
}
```

This should result in a layout that looks similar to Figure 5.25.

Three or More Columns

Creating layouts with three columns—or more, for that matter—isn't much different than creating two-column layouts.

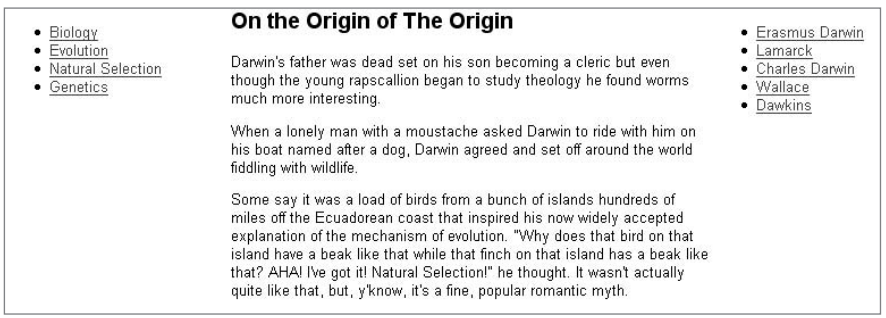

Biology
Evolution
Natural Selection
Genetics
On the Origin of The Origin
Darwin's father was dead set on his son becoming a cleric but even though the young rapscallion began to study theology he found worms much more interesting.
When a lonely man with a moustache asked Darwin to ride with him on his boat named after a dog, Darwin agreed and set off around the world fiddling with wildlife.
Some say it was a load of birds from a bunch of islands hundreds of miles off the Ecuadorean coast that inspired his now widely accepted explanation of the mechanism of evolution. "Why does that bird on that island have a beak like that while that finch on that island has a beak like that? AHA! I've got it! Natural Selection!" he thought. It wasn't actually quite like that, but, y'know, it's a fine, popular romantic myth.
Erasmus Darwin
Lamarck
Charles Darwin
Wallace
Dawkins

FIGURE 5.28 Three columns—content flanked by two navigation bars.

So, let's assume our HTML chunks are structured like this:

```
<div id="navigation1" class="nav">
     <!--stuff-->
</div>
<div id="navigation2" class="nav">
     <!--stuff-->
</div>
<div id="content">
     <!--stuff-->
</div>
```

Taking it one step at a time, positioning the first navigation chunk is just the same as in the two-column layout:

```
#navigation1 {
    position: absolute;
    left: 0;
    width: 200px;
}
```

And we can do almost the same thing with the second navigation chunk:

```
#navigation2 {
    position: absolute;
    right: 0;
    width: 150px;
}
```

Now, with the content area, all we need to do is squeeze it in on both sides, rather than just the left:

```
#content {
    margin: 0 150px 0 200px;
}
```

www.htmldog.com/examples/pagelayout4.html

If you don't want the navigation bars to flank the content but to stand next to each other instead, all you have to do is manipulate the various left, right, and margin declarations, for example:

```
#navigation1 {
    position: absolute;
    left: 0;
    width: 200px;
}
#navigation2 {
    position: absolute;
    left: 200px;
    width: 150px;
}
#content {
    margin-left: 350px;
}
```

Another approach to placing columns side by side would be to float the columns in exactly the same way as described for two columns.

These approaches can be used for as many columns as you like, but just keep in mind that there are still a lot of users out there whose screens are only 800 pixels wide.

Adding a Page Header

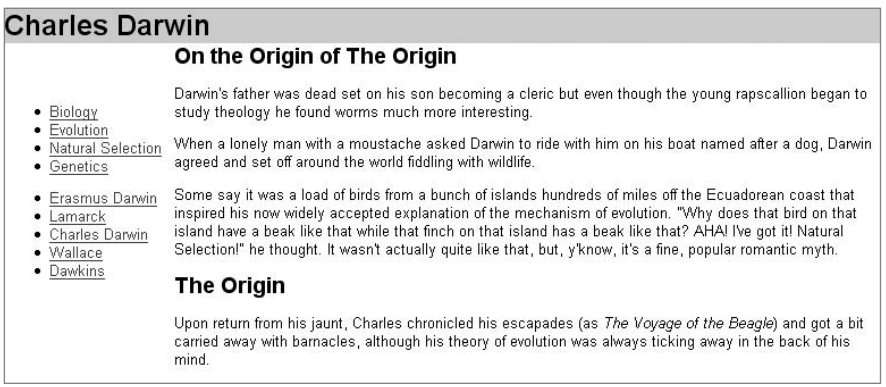

Charles Darwin
On the Origin of The Origin
Darwin's father was dead set on his son becoming a cleric but even though the young rapscallion began to study theology he found worms much more interesting.
When a lonely man with a moustache asked Darwin to ride with him on his boat named after a dog, Darwin agreed and set off around the world fiddling with wildlife.
Some say it was a load of birds from a bunch of islands hundreds of miles off the Ecuadorean coast that inspired his now widely accepted explanation of the mechanism of evolution. "Why does that bird on that island have a beak like that while that finch on that island has a beak like that? AHA! I've got it! Natural Selection!" he thought. It wasn't actually quite like that, but, y'know, it's a fine, popular romantic myth.
The Origin
Upon return from his jaunt, Charles chronicled his escapades (as The Voyage of the Beagle) and got a bit carried away with barnacles, although his theory of evolution was always ticking away in the back of his mind.
Biology
Evolution
Natural Selection
Genetics
Erasmus Darwin
Lamarck
Charles Darwin
Wallace
Dawkins

FIGURE 5.29 Popping a header on top of columns.

Adding a page header—space that might be used for branding and/or navigation—is easy. All you need to do is make sure that any absolutely positioned boxes are explicitly positioned below it. Continuing with the practice of separating specific chunks of the document with divs, we can work with the following HTML:

```
<div id="header">
    <!--stuff-->
</div>
<div id="navigation">
    <!--stuff-->
</div>
<div id="content">
    <!--stuff-->
</div>
```

In this two-column example, all that needs to be positioned is the navigation bar, because once this is taken out of the flow the content area will automatically sit

beneath the header, so you don't really need to do much different in the CSS (unless you want to):

```
#navigation {
    position: absolute;
    left: 0;
top: 100px;
    width: 150px;
}
#content {
    margin-left: 150px;
}
```

www.htmldog.com/examples/pagelayout5.html

Adding a Footer

Footers are a bit trickier than headers due to the nature of absolutely positioned boxes sitting on top of other non-positioned elements. Still, let's see what we can do by starting with the following HTML:

```
<div id="header">
    <!--stuff-->
</div>
<div id="navigation">
    <!--stuff-->
</div>
<div id="content">
    <!--stuff-->
</div>
<div id="footer">
    <!--stuff-->
</div>
```

If you can *guarantee* that the navigation area will be shorter than the content area then you have no worries. Applying similar CSS as before, you don't actually need to do anything special with the footer box because it will sit directly below the content automatically.

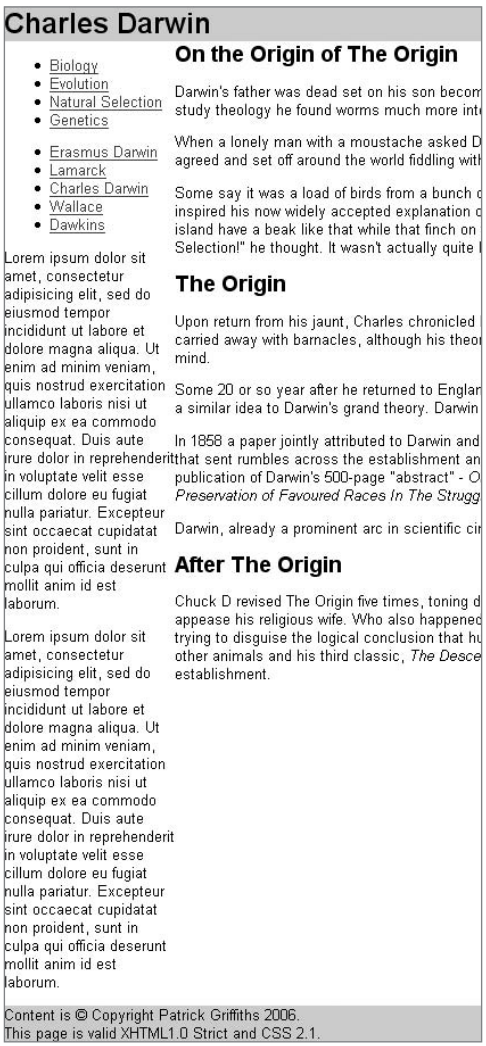

Charles Darwin
Biology
Evolution
Natural Selection
Genetics
Erasmus Darwin
Lamarck
Charles Darwin
Wallace
Dawkins
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
On the Origin of The Origin
Darwin's father was dead set on his son becom
study theology he found worms much more int
When a lonely man with a moustache asked D
agreed and set off around the world fiddling wit
Some say it was a load of birds from a bunch o
inspired his now widely accepted explanation o
island have a beak like that while that finch on
Selection!" he thought. It wasn't actually quite
The Origin
Upon return from his jaunt, Charles chronicled
carried away with barnacles, although his theo
mind.
Some 20 or so year after he returned to Englar
a similar idea to Darwin's grand theory. Darwin
In 1858 a paper jointly attributed to Darwin and
that sent rumbles across the establishment an
publication of Darwin's 500-page "abstract" - O
Preservation of Favoured Races In The Strugg
Darwin, already a prominent arc in scientific ci
After The Origin
Chuck D revised The Origin five times, toning d
appease his religious wife. Who also happened
trying to disguise the logical conclusion that hu
other animals and his third classic, The Desce
establishment.
Content is © Copyright Patrick Griffiths 2006.
This page is valid XHTML1.0 Strict and CSS 2.1.

FIGURE 5.30 Tucking in a footer.

More often than not, though, you can't *rely* on the navigation bar being shorter than the content area. As Figure 5.31 demonstrates, using the above method of incorporating a footer, if the navigation bar is longer than the content, because it is absolutely positioned it will lie on top of the static footer.

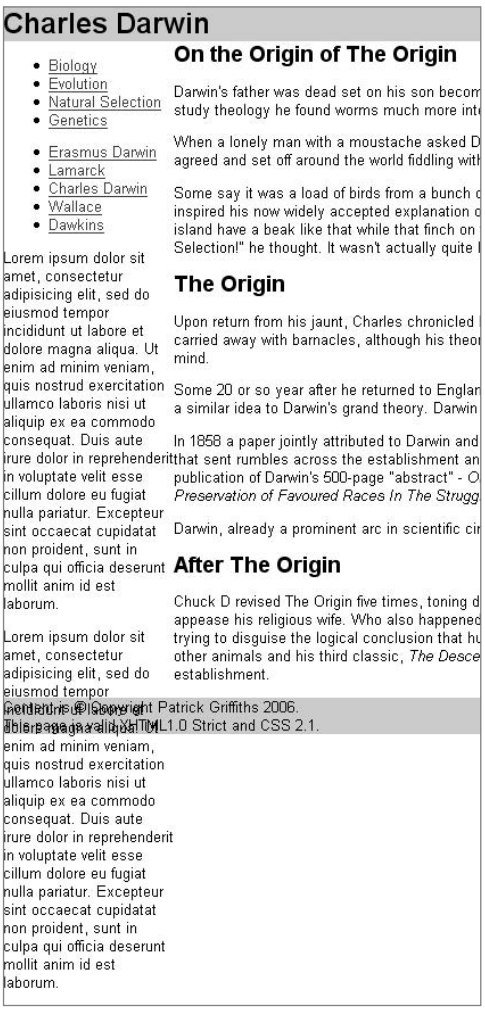


FIGURE 5.31 A static footer will fall underneath the navigation if the navigation is too long.

If you could rely on the navigation area always being longer than the content area, then you could absolutely position the content area instead, and leave the navigation area as the static box that the footer can sit under, but this is rarely a practical option.

www.htmldog.com/examples/pagelayout6.html

It is because of absolutely positioned boxes being pulled out of the normal flow of the page that it is impossible to predict where they will end (where the bottom of the box will be)—you can't place something on a page in relation to an absolute box. So, if you want a footer that works, a sensible option would be to go for floating columns and simply clear the footer of the floats:

```
#navigation {
    width: 10em;
    background: lime;
    float: left;
}
#content {
    margin-left: 10em;
}
#footer {
    clear: both;
}
```

If the content is longer than the navigation, then nothing will happen—the footer will happily just sit below it. If the navigation is longer, however, the footer will clear the float and sit underneath that instead.

www.htmldog.com/examples/pagelayout7.html

Putting It All Together

This example (shown in Figure 5.32) demonstrates most of the methods mentioned in this chapter, incorporating different padding, border, and margin situations as well as display types together with floated elements all inside a positioned layout.

The basic structure of the HTML looks something like this:

```
<div id="container">
<div id="header">
    <!--stuff-->
</div>
<div id="navigation">
    <!--stuff-->
</div>
```

```
<div id="content">
    <!--stuff-->
</div>
</div>
```

The header has a fixed height and a background image, with the `h1` element, which is necessary for the structured document, removed from sight with the `display: none` alternative.

```
#header {
    padding: 0.5em;
    background: url(images/charlesdarwin.gif);
}
```

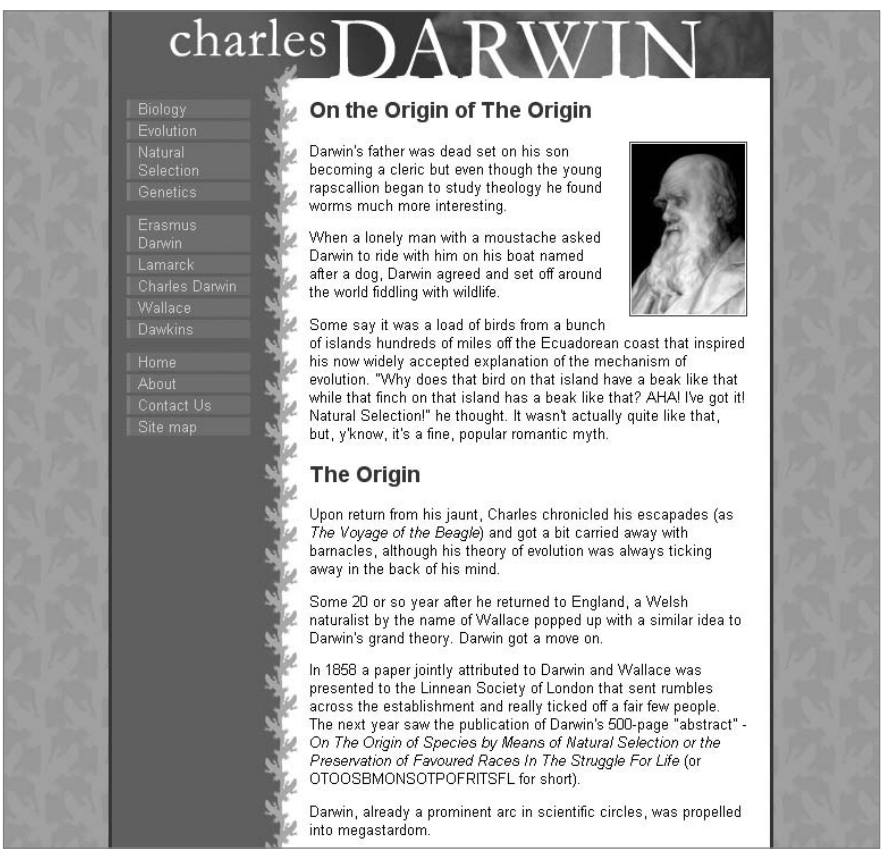

charles DARWIN
Biology
Evolution
Natural Selection
Genetics
Erasmus Darwin
Lamarck
Charles Darwin
Wallace
Dawkins
Home
About
Contact Us
Site map
On the Origin of The Origin
Darwin's father was dead set on his son becoming a cleric but even though the young rapscallion began to study theology he found worms much more interesting.
When a lonely man with a moustache asked Darwin to ride with him on his boat named after a dog, Darwin agreed and set off around the world fiddling with wildlife.
Some say it was a load of birds from a bunch of islands hundreds of miles off the Ecuadorean coast that inspired his now widely accepted explanation of the mechanism of evolution. "Why does that bird on that island have a beak like that while that finch on that island has a beak like that? AHA! I've got it! Natural Selection!" he thought. It wasn't actually quite like that, but, y'know, it's a fine, popular romantic myth.
The Origin
Upon return from his jaunt, Charles chronicled his escapades (as The Voyage of the Beagle) and got a bit carried away with barnacles, although his theory of evolution was always ticking away in the back of his mind.
Some 20 or so year after he returned to England, a Welsh naturalist by the name of Wallace popped up with a similar idea to Darwin's grand theory. Darwin got a move on.
In 1858 a paper jointly attributed to Darwin and Wallace was presented to the Linnean Society of London that sent rumbles across the establishment and really ticked off a fair few people. The next year saw the publication of Darwin's 500-page "abstract" - On The Origin of Species by Means of Natural Selection or the Preservation of Favoured Races In The Struggle For Life (or OTOOSBMONSOTPOFRITSFL for short).
Darwin, already a prominent arc in scientific circles, was propelled into megastardom.

FIGURE 5.32 Positioning, padding, borders, margins, and floating à go-go.

The navigation area is positioned to the left, underneath the header, with padding applied:

```
#navigation {
    position: absolute;
    top: 5em;
    left: 0;
    width: 7em;
    w\idth: 5em;
    background: #069;
    color: white;
    padding: 1em;
}
```

Note that because a width and padding are involved, the box model hack is needed to take care of IE 5.

Inside the navigation area, the links, which sit inside unordered lists (as is appropriate—see Chapter 6) have their `display` type set to `block` (they are `inline` by default) so that they will stretch to a uniform, "clickable" width.

There is also a "skip navigation" link (see Chapter 3) before this group of visible links, which, like the `h1` element, is removed from sight.

The content area itself provides the background color for the navigation bar with a thick left border, and has its own contents padded.

```
#content {
    padding: 1em;
    border-left: 7em solid #069;
}
```

Inside the content div, images are set to float to the right, with margins to the left and below to space them out from surrounding text. A 1px-wide border and padding create the subtle but effective outline.

Finally, the page is made a fixed width and centered:

```
#container {
    width: 600px;
    border: solid #900;
    border-width: 0 1px;
    margin: auto;
}
```

And there you have it.

www.htmldog.com/examples/darwin.html

The techniques explained in this chapter don't have to involve building blocks as specific as "headers" or "navigation bars." They can be applied to elements large and small, one after another or one inside another. The general principles are the same—it's all about shifting chunks around the page.

This page intentionally left blank

chapter 6

Lists

LISTS MIGHT NOT seem like that big a deal. Now and then they may pop up in your content, and you certainly need to take control of them, but even if your content is primarily made up of paragraphs and pictures lists should usually make their way onto a page because they are the best way to structure navigation.

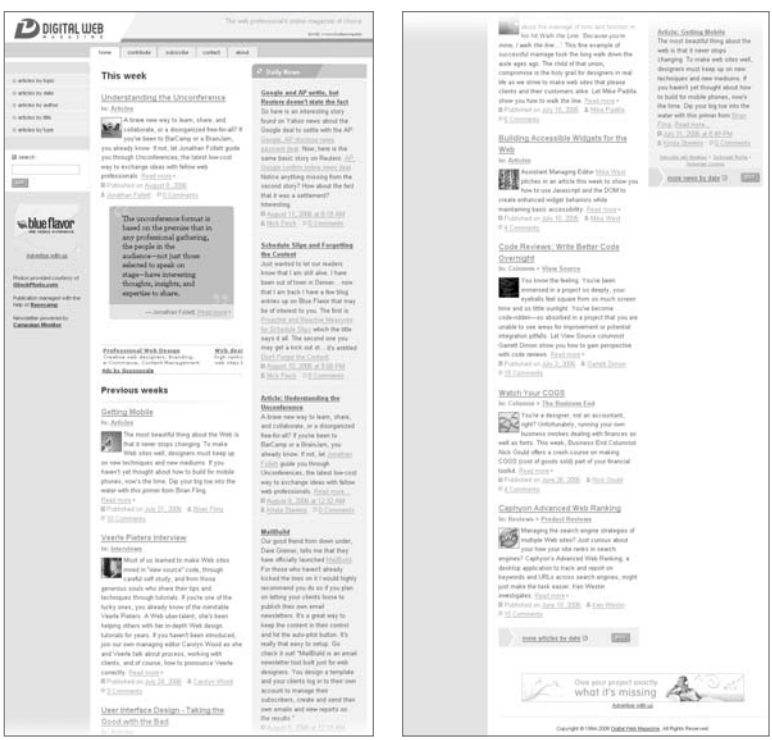


FIGURE 6.1 The illustrations in this chapter are taken from Digital Web Magazine (*digital-web.com*).

Structuring Lists

You should find that pretty much any list will fit into one of the three types available to you: an unordered list, an ordered list, or a definition list.

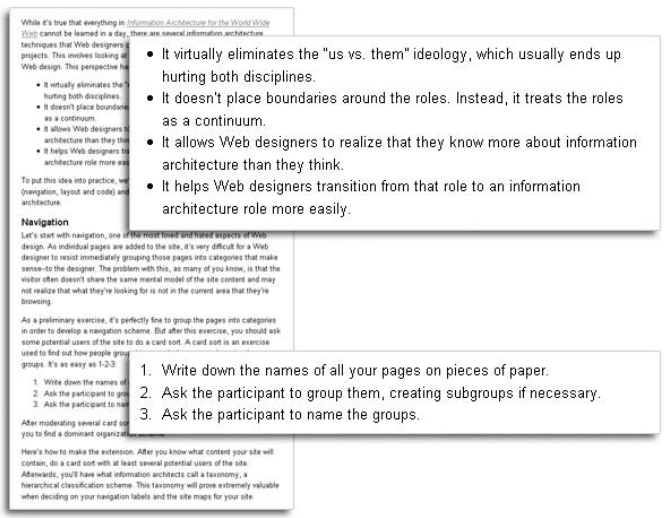

It virtually eliminates the "us vs. them" ideology, which usually ends up hurting both disciplines.
It doesn't place boundaries around the roles. Instead, it treats the roles as a continuum.
It allows Web designers to realize that they know more about information architecture than they think.
It helps Web designers transition from that role to an information architecture role more easily.
1. Write down the names of all your pages on pieces of paper.
2. Ask the participant to group them, creating subgroups if necessary.
3. Ask the participant to name the groups.

FIGURE 6.2 Straightforward unordered (top) and ordered (bottom) lists, with their default bullet and numbered styling, within an article by Joshua Kaufman.

Unordered and Ordered Lists

Unordered and ordered lists are your bog-standard list lists. `ul` defines an unordered list (for non-ordinal items, in which any item could feel just as at home at one point in the list as any other); `ol` defines an ordered list (in which each item is in some way lower or higher than the item before or after it); and in either list, list items are defined with `li`. Easy!

```
<ul>
    <li>This</li>
    <li>That</li>
    <li>The other</li>
</ul>
```

```
<ol>
    <li>The first thing</li>
    <li>The second thing</li>
    <li>The third thing</li>
</ol>
```

www.htmldog.com/examples/lists1.html

Nested Lists

You can use these same elements to create more complex, nested lists, as well—simply plonk another `ul` or `ol` element inside an `li` element:

```
<ul>
    <li>This
        <ul>
            <li>This type of this</li>
            <li>That type of this
                <ul>
                    <li>This type of that type of this</li>
                    <li>That type of that type of this</li>
                    <li>The other type of that type of this</li>
                </ul>
            </li>
            <li>The other type of this</li>
        </ul>
    </li>
    <li>That</li>
    <li>The other</li>
</ul>
```

www.htmldog.com/examples/lists2.html

11. Test, Test, Test
After you have finished the design of your landing page, test it with a small user group. Go over a checklist with your design team:
Is the whole page focused?
Does the message match the advertisement?
Have you reduced all distractions?
Is critical information above the fold?
Are there enough conversion exits?
Does the page enhance your brand?

FIGURE 6.3 Nested lists and more: In this Digital Web article by Michael Nguyen, the tail end of an ordered list sees paragraphs and an unordered list within the list items.

Definition Lists

Definition lists do away with `ul`, `ol`, and `li` elements. They use a separate set of elements to achieve a list that involves a combination of terms and descriptions for each "item": `dl` (the definition list), `dt` (a definition term), and `dd` (a definition description of the term).

```
<dl>
    <dt>Cat</dt>
    <dd>A little furry thing that purrs.</dd>
    <dt>Dog</dt>
    <dd>A big shaggy thing that barks.</dd>
</dl>
```

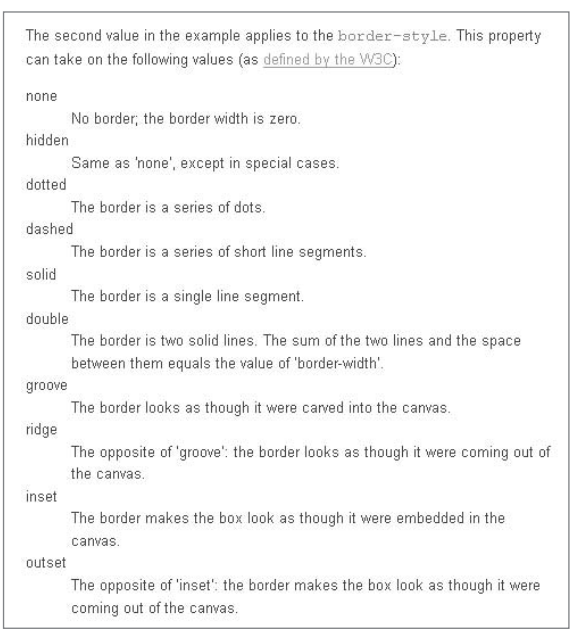
`dt` and `dd` always need to go together, but it doesn't matter which comes first and it doesn't matter how many you have for each item. You can have *x* terms followed by *y* descriptions, or *x* descriptions followed by *y* terms (depending on how you want your glossary to work, for example):

```
<dl>
    <dt>Cat</dt>
    <dd>Any member of the family Felidae.</dd>
    <dd>The domesticated species of that family, Felis silvestris.
</dd>
    <dd>A little furry thing that purrs.</dd>
```

```
    <dt>Dog</dt>
    <dt>Yo Momma</dt>
    <dd>A big shaggy thing that barks.</dd>
</dl>
```

www.htmldog.com/examples/lists3.html

The extent to which definition lists should be used has raised a few arguments. Whereas one school of thought believes you should only use them to define a list of explicit terms and definitions (something that I happen to agree with—we are talking about *semantics*, after all), there is another that claims it is just as valid to use them for any list where each set of items is related. One example (made directly by the W3C no less) is that of dialogue, in which the person speaking can be defined by the `dt` element and what they are saying by the `dd` element. Another would be a list of websites, whose names are marked up with `dt` and descriptions of them with `dd`.


The second value in the example applies to the border-style. This property can take on the following values (as defined by the W3C):

none
No border; the border width is zero.
hidden
Same as 'none', except in special cases.
dotted
The border is a series of dots.
dashed
The border is a series of short line segments.
solid
The border is a single line segment.
double
The border is two solid lines. The sum of the two lines and the space between them equals the value of 'border-width'.
groove
The border looks as though it were carved into the canvas.
ridge
The opposite of 'groove': the border looks as though it were coming out of the canvas.
inset
The border makes the box look as though it were embedded in the canvas.
outset
The opposite of 'inset': the border makes the box look as though it were coming out of the canvas.

FIGURE 6.4 Nested lists and more: In this Digital Web article by Michael Nguyen, the tail end of an ordered list sees paragraphs and an unordered list within the list items.

Lists as Navigation

Essentially, a navigation bar on a website is simply a list of links. So, accepting this, the most obvious choice for marking up a navigation area is a list element of some sort—usually an unordered list, or number of unordered lists.

This could take the form of a big nested list:

```
<ul>
    <li>Services
        <ul>
        <li>Peanut farming</li>
        <li>Hamster sitting</li>
        <li>Car valet</li>
        </ul>
    </li>
    <li>Products
        <ul>
        <li>Peanuts</li>
        <li>Flat hamsters</li>
        <li>Cars</li>
        </ul>
    </li>
    <li>Misc.
        <ul>
        <li>About us</li>
        <li>Contact us</li>
        <li>Site map</li>
        </ul>
    </li>
</ul>
```

Or it could take the form of a number of lists:

```
<h2>Services</h2>
<ul>
    <li>Peanut farming</li>
    <li>Hamster sitting</li>
```

```
        <li>Car valet</li>
    </ul>
    <h2>Products</h2>
    <ul>
        <li>Peanuts</li>
        <li>Flat hamsters</li>
        <li>Cars</li>
    </ul>
    <h2>Misc.</h2>
    <ul>
        <li>About us</li>
        <li>Contact us</li>
        <li>Site map</li>
    </ul>
```

Then again, some might argue it could even take the form of a definition list:

```
    <dl>
        <dt>Services</dt>
        <dd>Peanut farming</dd>
        <dd>Hamster sitting</dd>
        <dd>Car valet</dd>

        <dt>Products</dt>
        <dd>Peanuts</dd>
        <dd>Flat hamsters</dd>
        <dd>Cars</dd>

        <dt>Misc.</dt>
        <dd>About us</dd>
        <dd>Contact us</dd>
        <dd>Site map</dd>
    </dl>
```

As Chapter 5, "Layout," makes clear, lists in this situation will normally be surrounded by `div` tags or have an `id` applied to the opening list tag so that they can be specifically targeted by CSS without affecting other `ul`, `ol`, or `dl` elements in the web page.

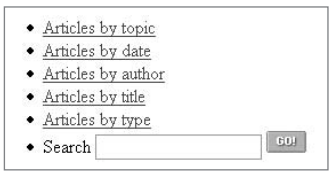


FIGURE 6.5 The secondary navigation in Digital Web Magazine is a simple list containing links and a form, shown here in a browser's default styles.

FIGURE 6.6 With the author CSS on top, the navigation is transformed into something quite spiffingly eye-catching.

Presenting Lists

Once the list structure is in place, you can bend and stretch every presentational aspect of it you like. Not only can you change the text, links, or images you might have inside the list items with the techniques discussed in Chapters 2, 3, and 4, you can also take the list by the scruff of the neck and alter list markers, margins, and even change the top-to-bottom vertical nature of the list to a left-to-right horizontal list.

List Markers—Bullets, Numbers, and Images

List markers in unordered and ordered lists can vary. These defaults tend to be the same from browser to browser, but you can change them to suit, choosing from filled-circle, square, or empty-circle bullets or numbers or Roman numerals.

The color and the size of the list item marker in `ul` and `ol` elements is taken from the `color` and `font-size` properties (see Chapter 2, "Text") of the list item.

The `list-style-type` property can be used to set the type of the list marker bullet or numbering system within a list. This can be applied to any (non-definition) list

regardless of whether it is ordered or unordered. These are some of the more practical values that can be used:

- `none`—No list marker. This can be handy when you want to present lists that don't appear in main content and don't need to stand out from the crowd with markers—as in navigation bars, for example.
- `disc`—solid circles
- `circle`—hollow circles
- `square`—solid squares
- `decimal` (which is default for `ol` elements)—1, 2, 3, 4, etc.
- `lower-roman`—i, ii, iii, iv, etc.
- `upper-roman`—I, II, III, IV, etc.

```
ol { list-style-type: lower-roman; }
ul { list-style-type: square; }
ul ul { list-style-type: circle; }
```

This example applies lower-roman numerals to ordered lists, square bullets to top-level unordered lists, and circular bullets to all unordered lists nested within unordered lists.

www.htmldog.com/examples/lists4.html

You can also provide something more customized by using `list-style-image`. This specifies an image to be used as the list marker for a list item. It can be used if you just don't like any of the `list-style-type` options and replaces the list item markers with your own custom-made wonders.

```
ul { list-style-image: url(images/arrow.gif); }
```

Note that another popular way of styling bullets is to apply small, non-repeating background images (see Chapter 4, "Images") and a bit of padding (see Chapter 5) to the left of each list item.

By default, lists will place the marker of each list item outside the content box, which means that when it comes to styling list items with backgrounds or borders, for example, the bullet will aloofly hang about outside. You can pull the marker

inside the content box to deal with such circumstances by setting the `list-style-position` property to `inside`.

```
ul { list-style-position: inside; }
```

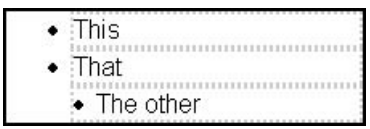


FIGURE 6.7 The third list item is set to `list-style-position: inside;` pulling the list marker inside the content area of the `li` box. The dark solid border shows the outline of the `ul` element and the light dotted borders show the outline of the `li` elements.

www.htmldog.com/examples/lists5.html

`list-style` is a shorthand property used to specify the style of a list item marker by combining `list-style-type`, `list-style-position`, and `list-style-image`.

Like all shorthand properties, this simply involves specifying one or more values that you would otherwise use in the longhand versions.

```
ul { list-style: none url(images/arrow.gif) inside; }
ul ul { list-style: disc outside; }
```

DOING AWAY WITH UNWANTED PADDING AND MARGINS

Although padding and margins are discussed in detail in Chapter 5, they are worth mentioning here because a browser will usually apply padding and margins to lists by default and it is quite possibly something that you will want to manipulate or even get rid of (especially if you are using a list for navigation).

A margin above and below lists will probably come as no surprise, but `ul` and `ol` elements will also have spacing to the left by default. IE will apply a margin and other browsers will apply padding. So if you want to get rid of this, simply apply:

```
ul {
    padding: 0;
    margin: 0;
}
```

`dd` elements also have a left margin by default, which can be eradicated in the same way if you so choose.

IE AND UNWANTED SPACES

When list items that contain other elements, such as links, are displayed inline, you may find IE inexplicably putting spacing between the list items. This is actually a space character that comes about from the browser incorrectly handling the white space within the code.

For example, if you have:

```
<ul>
    <li><a href="#">This</a></li>
    <li><a href="#">That</a></li>
    <li><a href="#">The other</a></li>
</ul>
```

And you apply `display: inline` to the list items, you will find that there are spaces after each item, which will often be unwanted, particularly if you want very tight control over the styling of the list.

Unfortunately, the only way to get around this is to arrange the HTML so that the list items are right next to one another, such as:

```
<ul><li><a href="#">This</a></li><li><a href="#">That</a></
li><li><a href="#">The other</a></li></ul>
```

If you want to maintain your indentations so that the code is more readable, you could try one of the following methods:

```
<ul>
    <li><a href="#">This</a></li><
    li><a href="#">That</a></li><
    li><a href="#">The other</a></li><
/ul>
```

or:

```
<ul>
    <li><a href="#">This</a></li><!--
    --><li><a href="#">That</a></li><!--
    --><li><a href="#">The other</a></li><!--
--></ul>
```

Horizontal Lists

FIGURE 6.8 Digital Web's site-wide primary navigation and the years depicted at the top of this page are unordered and ordered lists, respectively. The primary navigation (the tabs) is made up of list items set to float left. The list of years comprises list items set to display inline.

Chapter 5 explains how you can alter the way an element box behaves by using the `display` property. It doesn't take a great leap of imagination to figure out how to break the default vertical style of a list and present list items side by side, which you might want to do if you want a horizontal navigation bar. The default `display` of a list item in an unordered or ordered list is `list-item`, which itself is block-line in its rendering. Simply changing the `display` of `li` to `inline` will override this, and voila!—the list items line up horizontally as opposed to vertically.

```
li { display: inline; }
```

Floating each list item is an alternative method of achieving horizontal lists (once again, see Chapter 5 for more details on floating), and although it is slightly more complicated, it is also more versatile than the `display: inline` method because you can maintain block-level qualities such as manipulation of vertical padding and borders, etc. If you want to apply more complex styles to horizontal list items, this may be preferable.

To contain the list items you can also float the list itself, enabling you to manipulate its background/border properties, etc. As long as the element following the list is set to `clear` the float, you should find yourself with the basics of a smooth horizontal list.

```
ul, li { float: left; }
#afteralist { clear: left; }
```

TECHNIQUE: TABS

Shimmy over to *www.htmldog.com/articles/tabs/* for further insight, and a plethora of examples, about horizontal lists, and techniques for achieving that über-sexy way of presenting navigation—with tabs.

chapter 7

Scripts & Objects

IT JUST SITS THERE. It doesn't *do* anything. Well, that's kind of the point of HTML and CSS—it's just a way of structuring and presenting largely textual content. The whiz-bang-pop is the job of other languages and file types. Close to home you've got JavaScript, which allows you to dynamically manipulate the parts of an HTML page and then you've got your completely alien objects like videos and Flash files. They may not be a part of HTML or CSS, but they still rely on HTML to get them to work in a web page.

JavaScript and the DOM

JavaScript is a commonly used and widely supported scripting language that can be used to add interactive behaviors such as rollovers, form validation, and even switching between different style sheets. It can be applied to an HTML document with the `script` element or "event attributes" in individual tags. Through the Document Object Model (DOM), the W3C's standardized model for the structure of a web page, it is possible to manipulate any part of a web page with JavaScript.

The `script` Element

`script` defines a block of script, and is the tool of choice for inserting a chunk of JavaScript into an HTML page.

The script itself can be placed between the opening and closing `script` tags, like so:

```
<script type="text/javascript">
function satsuma() {
    alert("SAAAATSUUUUMAAAA!!!");}
</script>
```

Alternatively, a script can be kept in a separate file and applied like so:

```
<script type="text/javascript" src="kumquat.js"></script>
```

The `type` attribute is required, and will always have the value "`text/javascript`" when using JavaScript, and just like in an `img` tag, the `src` attribute points to the location of the external file.

To accommodate users who don't have JavaScript-enabled browsers, or those who choose to switch it off, you can provide alternative content by using a `noscript` element anywhere inside the `body` element. The content of this element will only show up when the browser can't detect JavaScript.

```
<noscript>
<p>What? No JavaScript? Well what am I supposed to do now? Can't you
get a new browser or something?</p>
</noscript>
```

Event Attributes

JavaScript code can be invoked when the user does something, such as clicking a button, rolling over an element, or loading a page. You can apply event attributes to just about any opening HTML tag (such as `onclick` in a submit button, `onmouseover` in a link, or `onload` in the `body` tag) that will pick up on such actions and when they take place, the value of the attribute, which would be a piece of JavaScript code, will be executed:

```
<a href="newfangled.html" onclick="alert('SAAAATSUUUUMAAAA!!!');">
DO IT!</a>
```

While the value of the attribute could contain all of the required JavaScript (such as that in the example above), it usually doesn't. The values of event attributes tend to make calls to functions defined in the script element (whether those functions are actually in the page or in an external file). This cuts down on the volume of inline code and makes common actions available in a central, reusable, location:

```
<a href="newfangled.html" onclick="satsuma();">DO IT!</a>
```

Having said all that, similar to the point made in Chapter 1, "Getting Started," about the `style` attribute, if you're taking JavaScript seriously it should be unobtrusive—HTML elements can be targeted through the DOM with JavaScript without the need of event attributes, which is a much nicer, easier way to manage, and more powerful way of doing things.

Manipulating the DOM

Put simply, the DOM is a standardized model of every part of a web page, including the HTML and CSS code, that is commonly manipulated with JavaScript.

The powerful ability to manipulate any and every part of any and every element on a page means that you can do away with event attributes altogether and separate out another layer: behavior, which carries similar benefits to separating structure and presentation. With the DOM you should be able to place all of your code inside a `script` element (be that in the page itself or accessed in a .js file) and dynamically remote-control the page.

This is the modern, cutting-edge way of using JavaScript. Like web-standard HTML and CSS, using DOM JavaScript leads to lighter, more manageable code. The philosophy and practice of DOM Scripting is a huge subject unto itself, and is somewhat outside the remit of this book. There are now many good quality books (see Figure 7.1) and online resources that delve right into it (*http://www.webstandards.org/action/dstf/* is a good starting point).

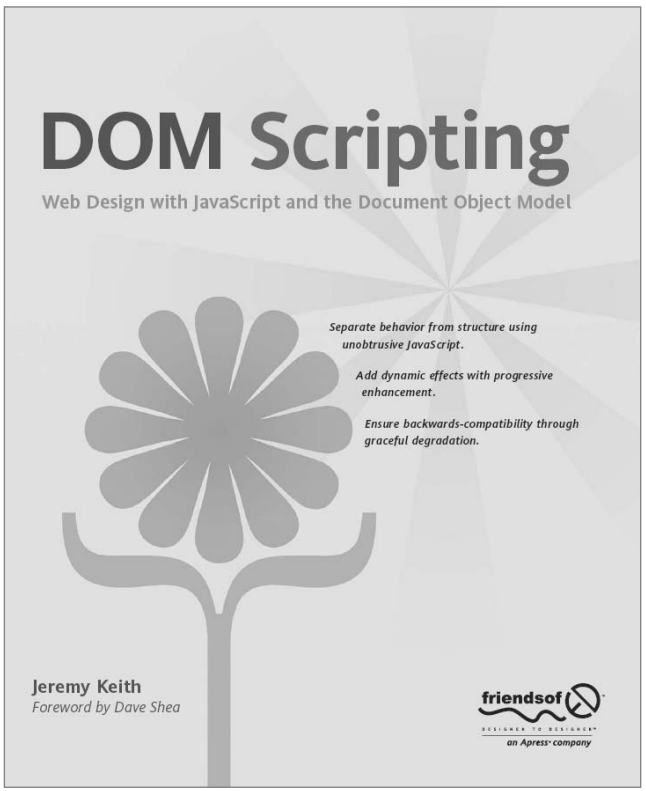

DOM Scripting
Web Design with JavaScript and the Document Object Model
Separate behavior from structure using unobtrusive JavaScript.
Add dynamic effects with progressive enhancement.
Ensure backwards-compatibility through graceful degradation.
Jeremy Keith
Foreword by Dave Shea
friendsof
DESIGNER TO DESIGNER
an Apress company

FIGURE 7.1 If you want to get to grips with best-practice JavaScript, once you're confident with your HTML and CSS, there are many good books out there, such as *DOM Scripting* by Jeremy Keith (Friends of Ed), which will give you a great introduction.

Objects

If you have a snazzy little file like an MPEG video or a Flash movie that you want to put it in your web page, you can "embed" such a foreign object with an `object` element.

Objects usually depend on some form of "plug-in"—a special piece of software that is added on to the browser (such as the Flash Player) so that the file can be deciphered and viewed (or heard).

The basic idea is quite a simple one: Inside the opening `object` tag, you use the `type` attribute to let the browser know what kind of object it is (and which plug-in to use), the `data` attribute to point the browser to the actual object file, and then inside the `object` element you pass parameters to the object-playing plug-in using `param` elements.

If the user does not have the plug-in required to execute the file in an object, you can provide alternative content that will be applied in the object's place. This can be an error message, or replacement image (or any chunk of HTML you choose).

```
<object type="blueberry/kumquat" data="whatever.kmq">
    <param name="tangy" value="true" />
    <param name="segments" value="9" />
    <p>You don't have the Kumquat plugin, so you won't get any
juice.</p>
</object>
```

So `object` defines the object, `param` passes parameters to the object, and the rest of the HTML works as alternative content. Easy.

There is a host of other attributes that lend more control over the object (check out Appendix A, "XHTML Reference," to find out more), but perhaps the most important thing to point out at this stage is that IT DOESN'T WORK.

EMBEDDING OBJECTS IN A WEB STANDARDS WAY

The most popular way of inserting a Flash movie in an HTML page is by using a rather ugly block of code vomited up by someone at Macromedia many moons ago. Not only is this notorious code ugly, it's completely invalid because it involves the use of the `embed` tag, which has never been a part of any standard.

The much simpler, more logical, valid, pleasing-to-the-eye, and ultimately correct method looks something like this:

```
<object type="application/x-shockwave-flash" data="whatever.swf">
</object>
```

But unfortunately it's not that easy. Using this sensible method, the nonsensical Internet Explorer will wait until the Flash movie has completely downloaded before playing it. This may be fine with small Flash movies, but with longer ones you'll probably want to take advantage of Flash's ability to stream the movie—to play it *while* it is downloading.

Dammit.

There are two less-than-perfect methods for getting around this problem. The first is known as "Flash Satay" (see *alistapart.com/articles/flashsatay*) and this involves using similar code to that above, but twiddling the Flash movie itself so that a small Flash movie is used to play the main, streaming movie.

The second method revolves around the fact that Internet Explorer requires information given by the `classid` and `codebase` attributes in the opening `object` tag to work properly. Unfortunately, by applying these to get Flash to work in IE, it breaks down in other browsers where the movie won't work. Hixie's method (*In. hixie.ch/?start=1081798064&count=1*) utilizes the strange IE "feature" of conditional comments, whereby Internet Explorer can be forced to ignore a chunk of HTML that displays the movie in other browsers:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=6,0,40,0">
    <param name="movie" value="whatever.swf">
    <!--[if !IE]> <-->
    <object type="application/x-shockwave-flash" data="whatever.
swf" ></object>
    <!--> <![endif]-->
</object>
```

There is no pretty way of embedding a Flash movie in HTML. The two methods mentioned above are standards-compliant, but they still require hacks, which should only be used when there's no other option. In this case, unfortunately, it seems there isn't.

As another example, Quicktime videos have the same kind of problem as Flash movies. You should be able to embed them in a page with this code:

```
<object type="video/quicktime" data="whatever.mov">
    <p>You aint got Quicktime.</p>
</object>
```

But once again, this straightforward code doesn't work in Internet Explorer and once more the code suggested by Quicktime's creator is daft `embed` tag nonsense.

To get it to work in IE you need something like this:

```
<object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
codebase="http://www.apple.com/qtactivex/qtplugin.cab">
    <param name="src" value="whatever.mov" />
</object>
```

But this won't work anywhere else.

The solution? Well, as with Flash, you could serve up different code to different browsers if you've got the server-side scripting skills or once more you could use Hixie's conditional comments. You can even use Flash Satay to display the Quicktime movie through a Flash movie…

Objects truly are a cross-compatibility headache.

This page intentionally left blank

chapter 8

Tables

TABLES ARE INFAMOUS in the web standards world. At the slightest whisper of their name, web standards aficionados have been known to experience involuntarily muscle spasms and bouts of uncontrollable cursing. The table's bad reputation comes from its prolific use as a means for laying out web pages—just a short casual web browse will reveal that most pages on the web have tables all over the place.

eventWax
The Easier, Smarter Way To Organize Special Events, From Conferences And Workshops To Parties, Gigs, And Receptions.
Take the Tour!
Lubricate Your Event!
Getting an account is quick, easy, and free. Give it a whirl!
Account name
Account address: http://accountname.eventwax.com
Email address
Password
Confirm Password
There are a few terms & conditions
Yep! I have read and agree to the terms & conditions
Create Account
Features: Customizable, hosted web site for your event Alternatively, hook up the registration pages with your own site Accept payments through your PayPal account Quick, searchable, sortable access to attendee details Keep track of activity through RSS feeds Comprehensive help, accessible externally, or within your admin area Dynamic mailing lists that can be targeted at attendees, or groups of attendees Export data as spreadsheets or XML
Coming Soon: Full, remote control and access with a feature-rich API Extensive integration with Upcoming.org Send out invites and collect RSVPs Multiple payment acceptance options, including invoicing and popular payment gateways such as Authorize.Net and Secpay Themed event sites with comments, attendee profiles, and other new interactive options
Event Wax is brewed and canned at the Vivabit factory. It has not been tested on non-human animals.
Questions? Comments? Please contact us at info@eventwax.com

FIGURE 8.1 The illustrations in this chapter are taken from Event Wax (*www.eventwax.com*).

They're not the best choice for layout—CSS is (see Chapter 5, "Layout"), but they're not entirely evil. A common mistake is believing that tables have no place on Planet Web Standards, but they do, in a slightly more modest role than page layout, but a much more sensible one for them: structuring and presenting genuine tabular data.

This is the place where you'll get to know how to do just that—from constructing basic data tables through to accessibility considerations and specific methods of styling them.

Basic Tables

Big, complex tables can get quite complicated to code, but they follow very logical structural rules. To create a basic table, all you need to do is establish a `table` element, then fill it with table rows (`tr`), and then fill them with cells of table data (`td`).

So let's start with the rows at first. Here's the beginnings of a table with three rows:

```
<table>
    <tr></tr>
    <tr></tr>
    <tr></tr>
</table>
```

You can't have rows without columns, though—it would all be just too one-dimensional. Although we don't define the columns explicitly, we can define each cell in the row, using `td` elements:

```
<table>
    <tr>
        <td>Cats</td>
        <td>Dogs</td>
        <td>Lemurs</td>
    </tr>
    <tr>
        <td>Tiger</td>
        <td>Grey wolf</td>
```

```
            <td>Indri</td>
        </tr>
        <tr>
            <td>Cheetah</td>
            <td>Cape hunting dog</td>
            <td>Sifaka</td>
        </tr>
    </table>
```

So here we have a table with three rows with three cells in each row, making it a 3×3 table. Capisce?

www.htmldog.com/examples/basictable.html

Now let's make this example a little bit more meaningful. Because "Cats," "Dogs," and "Lemurs" are actually headers of their respective columns, we can change them from `td` elements into `th` elements. It's still a cell, it still works pretty much the same, but the essential difference is that rather than your bog-standard table *data* cell, it's a table *header* cell. So all we would need to do is change that first row to:

```
<tr>
    <th>Cats</th>
    <th>Dogs</th>
    <th>Lemurs</th>
</tr>
```

Table header cells can also be used as headers for rows. For example, the table could be turned around the other way, and be structured like this:

```
<table>
    <tr>
        <th>Cats</th>
        <td>Tiger</td>
        <td>Cheetah</td>
    </tr>
    <tr>
        <th>Dogs</th>
        <td>Grey wolf</td>
        <td>Cape hunting dog</td>
```

```
        </tr>
        <tr>
            <th>Lemurs</th>
            <td>Indri</td>
            <td>Sifaka</td>
        </tr>
    </table>
```

www.htmldog.com/examples/headercells.html

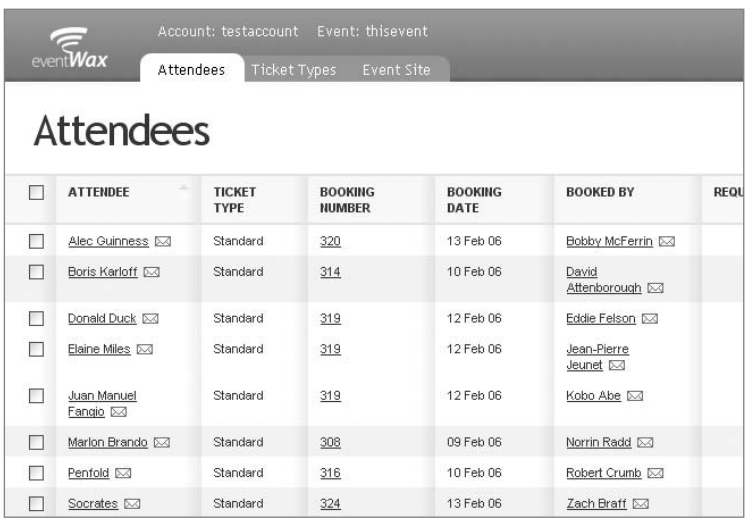


FIGURE 8.2 All dolled up with CSS, but below the surface is a straightforward table structure, with `table`, `tr`, `th`, and `td` elements.

Merging Cells

Not every row has to have the same number of cells in it and neither does every column. By manipulating the `rowspan` and `colspan` attributes inside the opening `td` or `th` tags, you can make those cells cover more than one row or column.

For example, if we wanted a higher classification than "Cats," "Dogs," and "Lemurs," we might have a slightly different top row:

```
<table>
    <tr>
        <th colspan="2">Carnivores</th>
        <th>Primates</th>
    </tr>
    <tr>
        <td>Tiger</td>
        <td>Grey Wolf</td>
        <td>Indri</td>
    </tr>
<!-- etc. -->
</table>
```

The first `th` element (with the content "Carnivores") will span the first two columns, leaving the third for the second `th` element ("Primates"). Because the three columns are covered, there is no need for a third `th` element.

www.htmldog.com/examples/colspan.html

Similarly, `rowspan` will cause a cell to spill over any number of rows:

```
<table>
    <tr>
        <th rowspan="2">Carnivores</th>
        <td>Tiger</td>
        <td>Cheetah</td>
    </tr>
    <tr>
        <td>Grey Wolf</td>
        <td>Cape hunting dog</td>
    </tr>
    <tr>
        <th>Primates</th>
        <td>Indri</td>
        <td>Sifaka</td>
```

```
        </tr>
    </table>
```

As the first `th` element in this example spans two rows, the second `tr` element contains two rather than three `td` elements because the first column of that row is already taken care of.

www.htmldog.com/examples/rowspan.html

Carnivores	Tiger	Cheetah	Caracal	Wildcat
	Grey Wolf	Cape hunting dog	Red fox	Fennec
Primates	Indri	Sifaka	Brown lemur	Dwarf lemur

Carnivores		**Primates**
Tiger	Grey Wolf	Indri
Cheetah	Cape hunting dog	Sifaka
Caracal	Red fox	Brown lemur
Wildcat	Fennec	Dwarf lemur

FIGURE 8.3 A bare-bones example, demonstrating the affects of `rowspan` and `colspan`.

Combinations of row- and column-spanned cells can lead to very complex tables and the bigger the table gets, the more difficult it can be to keep track of what should go where. It's often handy to work out exactly how you want the table structured beforehand (I have to admit to drawing such tables on a piece of paper first so that I can more easily figure out which cell needs to do what, where).

Captions

You can slap a caption on a table by using a caption element. This should be placed directly after the opening `table` tag and will be displayed above the table by default:

```
<table>
    <caption>Animal groups</caption>
    <!-- etc. -->
</table>
```

CAPTION POSITIONING

You can position the caption with the `caption-side` CSS property. Applying this to the `table` element (not the `caption` element) dictates on which side of the table the caption should be placed. See *www.htmldog.com/examples/colgroup.html* for an example, which also demonstrates, unfortunately, that `caption-side` isn't supported by Internet Explorer 6.

Values can be `top` (default), `right`, `bottom`, and `left`.

Grouping Rows

You can group together rows and split a table into a header, footer, and body by organizing rows into `thead`, `tfoot`, and `tbody` elements.

When tables are in some way broken, this should allow table parts to be repeated. When large tables are printed and take up more than one page, for example, the header and footer should appear on every printed page. Unfortunately, this isn't the case with Internet Explorer (which will just print them at the top and the bottom of the whole table), but is a nice feature with other, more compliant browsers.

Grouping rows can also provide a handy block to latch CSS on to (if you wanted to change the background color of a block of rows in a table, for example), and can aid accessibility, giving divisions of code for users to jump between.

These elements must be defined in the order `thead` > `tfoot` > `tbody` and not `thead` > `tbody` > `tfoot`. Don't worry—the final result will still have the `tbody` element sandwiched in between the header and the footer.

You can, if you want, have more than one `tbody` element, but you can only have one `thead` and `tfoot`.

```
<table>
<thead>
    <tr>
```

```
            <td>Header 1</td>
            <td>Header 2</td>
            <td>Header 3</td>
        </tr>
    </thead>
    <tfoot>
        <tr>
            <td>Footer 1</td>
            <td>Footer 2</td>
            <td>Footer 3</td>
        </tr>
    </tfoot>
    <tbody>
        <tr>
            <td>Cell 1</td>
            <td>Cell 2</td>
            <td>Cell 3</td>
        </tr>
        <!-- etc. -->
    </tbody>
    </table>
```

Targeting Columns

Although tables are built row by row, you can target columns with the `colgroup` and `col` elements, allowing you to apply attributes, such as a class, to all of the cells in a column or groups of columns.

`colgroup` allows attributes to be applied to set of columns and can be used on its own, along with the `span` attribute (in a similar way to using `rowspan` and `colspan` in `td` and `th` tags), to group the first *x* columns.

```
    <table>
        <colgroup span="2" class="alternative"></colgroup>
        <tr>
            <th>Cats</th>
            <th>Dogs</th>
            <th>Lemurs</th>
```

```
        </tr>
        <!-- etc. -->
    </table>
```

This example will, essentially, apply the "alternative" class to the first two columns.

Alternatively, `colgroup` can be used with `col` elements to focus on individual columns.

```
    <table>
        <colgroup>
            <col />
            <col class="alternative" />
            <col />
        </colgroup>
        <tr>
            <th>Cats</th>
            <th>Dogs</th>
            <th>Lemurs</th>
        </tr>
        <!-- etc. -->
    </table>
```

Here, the styles of the class "alternative" will be applied to the second column, i.e., the second cell in every row.

www.htmldog.com/examples/colgroup.html

You can also use the `span` attribute with `col` elements, and could, for example, apply them like this:

```
    <table>
        <colgroup>
            <col />
            <col span="2" class="alternative" />
        </colgroup>
        <!-- etc. -->
    </table>
```

Oh, but there had to be a catch, didn't there? Here it is: The only styles you can *validly* apply to columns are backgrounds (see Chapter 4, "Images"), borders, width, and visibility (Chapter 5).

In a strange twist of fate, Internet Explorer *appears* to behave much better than other browsers because it applies pretty much any CSS property to columns via `col` and `colgroup` elements, but, as it turns out, this is only because it acts in a mad wacky way. For a detailed explanation of this peculiar anomaly, go to this catchy web address to let Ian Hixon explain: *In.hixie.ch/?start=1070385285&count=1*.

Accessibility Considerations with Tables

If you follow the methods mentioned so far with content that is sensible tabular data, you should be well on your way to creating accessible tables. The major problem in terms of accessibility, however, is the two-dimensional nature of tables: You have rows and you have columns. Your eyes can see vertical and horizontal associations with little problem, but if you had to rely on your ears—if the table became linearized and were read out to you cell by cell by a screen-reader—it could get very confusing. Listening to a ream of numbers, completely out of context, for example, would not be very helpful.

Summaries

A quick and easy accessibility consideration is to always apply a *summary* to the table. This can be specified through the use of the `summary` attribute in the opening `table` tag.

```
<table summary="A brief overview of animals belonging to certain
taxonomic groups">
    <caption>Animal groups</caption>
    <!-- etc. -->
</table>
```

The value of `summary` won't be displayed, but it will be recognized—and read out—by screen-readers. This brief description of what's going on can make the gist of the table content much easier and quicker to understand, or completely ignore if it isn't of interest.

Associating Headers to Cells

With a summary, the user can get an idea of what to expect. But this doesn't solve the problem of tables becoming linearized and cells being taken out of their context when a screen-reader comes to tackle a table. Explicit associations between the cells and their headers can aid this process, allowing the row or column heading to be read out along with the data itself, giving the visually impaired user the context that a visually able user has.

By using the `scope` attribute within a header cell you can explicitly define what the header cell is a header for. The value of this attribute can be `row`, `col`, `rowgroup` (for `thead`, `tfoot`, and `tbody` elements), or `colgroup`.

```
<table>
    <tr>
        <th scope="col">Cats</th>
        <th scope="col">Dogs</th>
        <th scope="col">Lemurs</th>
    </tr>
    <tr>
        <td>Tiger</td>
        <td>Grey Wolf</td>
        <td>Indri</td>
    </tr>
    <!-- etc. -->
</table>
```

Associating Cells to Headers

Doing things the other way around from `scope`, the `headers` attribute can be used within a `td` or `th` tag to specify which cell or cells should be regarded as headers for it. The value can be a single ID name or a list of IDs separated by spaces.

```
<table>
    <tr>
        <th colspan="2" id="carnivores">Carnivores</th>
        <th id="primates">Primates</th>
    </tr>
```

```
    <tr>
        <th id="cats" headers="carnivores">Cats</th>
        <th id="dogs" headers="carnivores">Dogs</th>
        <th id="lemurs" headers="primates">Lemurs</th>
    </tr>
    <tr>
        <td headers="carnivores cats">Tiger</td>
        <td headers="carnivores dogs">Grey Wolf</td>
        <td headers="primates lemurs">Indri</td>
    </tr>
    <!-- etc. -->
</table>
```

By doing this, when reading out the table data, a screen-reader should first read out the data in the related header cell. For example, when it comes to the first `td` element in the example above, it should read out "Carnivores: Cats: Tiger."

You may not want long headers to be repeated every time a data cell for that header is read out and you can avoid this happening by supplying a shortened version of the header with the `abbr` (abbreviation) attribute:

```
<table>
    <tr>
        <th id="cats" abbr="Cats">Felidae - the cats</th>
        <th id="dogs" abbr="Dogs">Canidae  - the dogs</th>
        <th id="lemurs" abbr="Lemurs">Lemuridae - the lemurs</th>
    </tr>
    <tr>
        <td headers="cats">Tiger</td>
        <!-- etc. -->
    </tr>
    <tr>
        <td headers="cats">Cheetah</td>
        <!-- etc. -->
    </tr>
    <!-- etc. -->
</table>
```

On the first encounter with a data cell linked to a header, the whole header will be read out, such as "Felidae—the cats: Tiger" but on every subsequent pass, only the abbreviated form of the header will be read, such as "Cats: Cheetah."

Presenting Tables

Table cells can be styled just like any other content. Colors, backgrounds, font-size, text-align can all be applied, for example (see Chapter 2, "Text"), as can widths and padding (see Chapter 5). You can target the table, row, row group, column (although remember the limitations, as noted), or cells. For example:

```
td {
    text-align: center;
    vertical-align: middle;
    padding: 0.1em 1em;
}
col.alternative {
    background-color: #ddf;
}
```

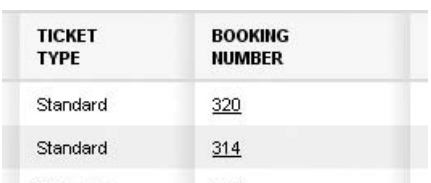


FIGURE 8.4 The tables in Event Wax use background images in each cell for the shadow effect, a hint of `border`, and a soupcon of `vertical-align`

There are also some table-specific CSS properties, though, that deal with table and cell borders, layout style, and what happens with empty cells.

Border Collapsing

Borders in tables are a little more complicated than your average box (see Chapter 5). Applying the `border` property to a table element will simply draw a four-sided border around the table's edge, rather than around the cells. To have a grid-like border

throughout the table and surrounding the cells you need to apply the border property to the cells themselves:

```
td { border: 1px solid black }
```

The results of this may not be exactly what you want, however, since each `td` element becomes a clearly defined individual box, rather than a cell within a grid. This is because the browser is using the "separated borders model," which completely separates cells, spacing them out from one another. You can change the border model, however, with the `border-collapse` property, which can be used to achieve an often-preferable alternative:

```
table { border-collapse: collapse }
```

This will invoke the "collapsing borders model," whereby cells share adjacent borders. All of the cells are pushed together and, quite cleverly, instead of pushing the borders up against each other, they "collapse" (much like margin collapsing—see Chapter 5), leaving only the wider of the two adjacent borders visible.

www.htmldog.com/examples/bordercollapse1.html

In the separated-borders model, theoretically you should be able to adjust the spacing between cells using the `border-spacing` property with the `table` element (such as `table { border-spacing: 1px; }`). Why *theoretically*? You guessed it: It isn't supported by Internet Explorer.

Collapsing will also occur when a table border comes into contact with cell borders. If the table border is narrower than the adjacent cell borders, then the table border should collapse, with the cell borders taking precedence. In Internet Explorer, though, the cell borders will always collapse, even if they are wider than the table border.

For example, if you had:

```
table {
    border-collapse: collapse;
    border: 1px solid black;
}
td {
    border: 10px solid #ccc;
}
```

You shouldn't be able to see the black table border because it should collapse. In IE, though, the 1px table border remains, and the adjacent cells have no adjacent borders (compare *www.htmldog.com/examples/bordercollapse2.html* in Firefox and IE, for example).

FIGURE 8.5 Without `border-collapse` to annihilate the spacing between cells and the limited support of `border-spacing`, the desired style would come up against a few problems.

Speedier Tables: the Fixed Layout Algorithm

Tables aren't the easiest of things for a browser to render. Your average table needs quite a few calculations—the browser must first go through the table, assessing the widths of every cell so that it can calculate the widths of columns and the table itself. Only after that will the table be drawn, with column widths optimized so that those with longer content on average will be wider.

The theory goes that for large, or numerous, tables, this automatic table layout algorithm can take a long time. In practice, with the cheetah speed of modern browsers, you're rarely going to come across a table where this is noticeable. You can, however, use the `table-layout` property to force the browser to use the "fixed layout algorithm" to speed things up.

Rather than going through the whole table and analyzing the content of all of the cells, this just takes a quick peek to see if there are any explicit widths applied to `col` elements or cells in the first row and then gets on with drawing the table.

Because this algorithm can't work out the width of the table, this should be explicitly specified also. Otherwise some browsers will ignore the `table-layout` declaration completely and use the automatic table layout algorithm to determine the width of the table. Interestingly, Internet Explorer will apply a width of 100% if none is specified.

The widths of columns are determined by the explicit width of `col` elements or, if none is specified, the explicit width of `td` (or `th`) elements in the first row. Cell widths in subsequent rows will be ignored.

Those columns that don't have an explicit width specified this way will share the rest of the width of the table equally. So if no widths are defined at all, all columns in the table will be an equal width.

`table-layout` isn't supported by IE 5.0, but that doesn't really matter because where this isn't supported, the table will still be rendered, it will just take longer.

```
table {
    table-layout: fixed;
    width: 100%;
}
```

www.htmldog.com/examples/tablelayout1.html

www.htmldog.com/examples/tablelayout2.html

Empty Cells

Empty cells (such as `<td></td>`, with no content in between the opening and closing tags) are an odd thing. In the collapsing borders model all is fixed and predictable: The cell is shown, it just won't have anything in it. With the separated borders model, however, the cells can either remain visible or can be hidden. By default, Internet Explorer hides empty cells (although it oddly decides to retain any applied backgrounds). By contrast, other browsers will show the empty cells by default, but you can opt to hide them with the `empty-cells: hide` declaration (which will hide everything, including any backgrounds).

`empty-cells: show` does the opposite, but IE won't take any notice so you're stuck with empty cells being hidden. You can get around this by putting any content in the cells, such as a non-breaking space (`<td> </td>`), which effectively makes it a no-longer-empty cell and so it will be shown in its full glory.

www.htmldog.com/examples/emptycells.html

So, in conclusion, if you want to hide empty cells, just apply `empty-cells: hidden` to take care of browsers other than IE (which hides them anyway). If you want to show empty cells, simply drop an ` ` character in each of them.

chapter 9

Forms

IF MONEY MAKES the world go around, then forms make the web go around. They are key to most commercial websites, which rely on taking personal information and credit card details. But they're handy for less capitalistic purposes too. A basic form can also be used to allow a user to submit a comment or question via a web page, or for gathering countless other types of useful information.

Forms are sometimes used in conjunction with client-side scripts for web application functionality (such as devising a simple calculator, for example), but are most commonly used as they were intended—to send data across the Internet.

What goes on after a form is submitted is a world beyond HTML and CSS, involving such alien server-side programming languages as PHP, ASP, or Perl that take the form data and do whatever needs to be done with it.

On the HTML and CSS side, all we have to do is make sure that the form itself is designed properly so that the necessary data is sent where it needs to go.

The basics are simple enough: You have a `form` element and within it you have a whole bunch of form fields and a submit button. The user fills in the fields, hits the button, and the data is sent.

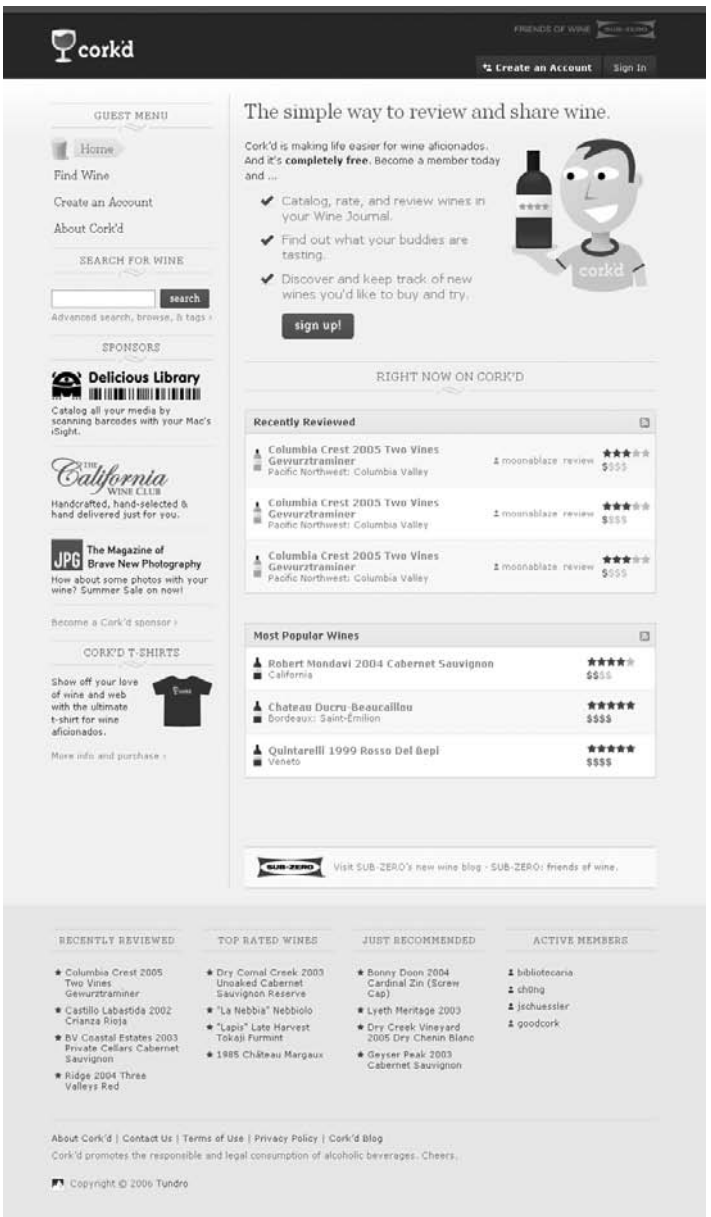

corkd
Create an Account
Sign In
GUEST MENU
Home
Find Wine
Create an Account
About Cork'd
SEARCH FOR WINE
search
Advanced search, browse, & tags ›
SPONSORS
Delicious Library
Catalog all your media by scanning barcodes with your Mac's iSight.
The California Wine Club
Handcrafted, hand-selected & hand delivered just for you.
JPG The Magazine of Brave New Photography
How about some photos with your wine? Summer Sale on now!
Become a Cork'd sponsor ›
CORK'D T-SHIRTS
Show off your love of wine and web with the ultimate t-shirt for wine aficionados.
More info and purchase ›
The simple way to review and share wine.
Cork'd is making life easier for wine aficionados. And it's completely free. Become a member today and ...
Catalog, rate, and review wines in your Wine Journal.
Find out what your buddies are tasting.
Discover and keep track of new wines you'd like to buy and try.
sign up!
RIGHT NOW ON CORK'D
Recently Reviewed
Columbia Crest 2005 Two Vines Gewurztraminer
Pacific Northwest: Columbia Valley
moonablaze reviews
$$$$
Columbia Crest 2005 Two Vines Gewurztraminer
Pacific Northwest: Columbia Valley
moonablaze reviews
$$$$
Columbia Crest 2005 Two Vines Gewurztraminer
Pacific Northwest: Columbia Valley
moonablaze reviews
$$$$
Most Popular Wines
Robert Mondavi 2004 Cabernet Sauvignon
California
$$$$
Chateau Ducru-Beaucaillou
Bordeaux: Saint-Émilion
$$$$
Quintarelli 1999 Rosso Del Bepi
Veneto
$$$$
SUB-ZERO
Visit SUB-ZERO's new wine blog - SUB-ZERO: friends of wine.
RECENTLY REVIEWED
Columbia Crest 2005 Two Vines Gewurztraminer
Castillo Labastida 2002 Crianza Rioja
BV Coastal Estates 2003 Private Cellars Cabernet Sauvignon
Ridge 2004 Three Valleys Red
TOP RATED WINES
Dry Comal Creek 2003 Unoaked Cabernet Sauvignon Reserve
"La Nebbia" Nebbiolo
"Lapis" Late Harvest Tokaji Furmint
1985 Château Margaux
JUST RECOMMENDED
Bonny Doon 2004 Cardinal Zin (Screw Cap)
Lyeth Meritage 2003
Dry Creek Vineyard 2005 Dry Chenin Blanc
Geyser Peak 2003 Cabernet Sauvignon
ACTIVE MEMBERS
bibliotecaria
ch0ng
jschuessler
goodcork
About Cork'd | Contact Us | Terms of Use | Privacy Policy | Cork'd Blog
Cork'd promotes the responsible and legal consumption of alcoholic beverages. Cheers.
Copyright © 2006 Tundra

FIGURE 9.1 The illustrations in this chapter are taken from the Cork'd website (*corkd.com*).

form Elements

Guess what a form element does. That's right! It defines a form. It is between the opening and closing `form` tags that all of the form fields, buttons, and other bits and bobs go.

```
<form action="processor.php" method="post">
<!-- a whole load of form fields -->
</form>
```

The opening form tag has two main attributes: `action` and `method`.

The value of the required `action` attribute tells the browser where to send the form data when it is submitted. This can be any URI, naming the location of the script (or page containing the script) that will process the form data.

The value of the `method` attribute tells the browser how to send the form data. You have two options here: `get` or `post`.

The `get` setting bolts the values of the form fields on to the URI supplied by the `action` attribute. So effectively, when the form is submitted, the user is taken to a very specific URI, which would look something like this:

http://www.whatever.com/processor.php?book=nineteen-eighty four&author=george orwell&datepublished=1949

The specific purpose of `get` is to *read* something—to retrieve data from somewhere, dependent on the data sent. An advantage of this method is that such a URI can be bookmarked or added to your browser's "favorites" list or shared with others (such as sent via email or instant messaging) because linking to a URI such as that above will have the same effect as inputting the data into the form and submitting it.

A disadvantage is that it isn't very secure. All the form values are there for everyone to see and it is also open to direct manipulation by the user. So whereas this method might be useful for locating a book in a catalog (hence retrieving data), for example, it wouldn't be suitable for sending credit card details or personal information.

Because `method="get"` is the default setting, you don't need to specify this attribute if this is the method you choose. So in fact, you'll probably only ever want to use it in the form of `method="post"`.

Instead of making the values part of a URI, the `post` method will send the form data as HTTP headers—pieces of information that are sent along with the URI, hidden away in the ether where they are invisible to all but the form-processing script.

Whereas `get` is used for reading something, `post` has the specific purpose of *writing* something (to a database, for example). The advantages and disadvantages are basically the opposite of the `get` method: Because the form data is not part of the URI, the form-results page cannot be bookmarked or shared with others. But because of this, it is also slightly more secure, and non-tinkerable, than the `get` method.

Form Fields and Buttons: `input`, `textarea`, and `select`

The form fields, where the user inputs data, come in a multitude of guises—text boxes, radio buttons, drop-down lists, to name a few—but they comprise just three elements: `input`, `textarea`, and `select`.

The `input` element is the 10-headed hydra of the trio, creating a different form control depending on the value of its `type` attribute. The other two, `textarea` and `select`, create just one control type each. These elements will be looked at in independent detail in a minute, but there are a few characteristics common to all three that we need to think about first.

The `name` Attribute

If all the inputted data in a form was sent without anything to identify each piece of data, a form-processing script wouldn't know what piece of data is what. "Things Fall Apart" or "Chinua Achebe" aren't helpful on their own, for example. What is needed is name/value pairs such as "book=Things Fall Apart" and "author=Chinua Achebe". The `name` attribute supplies this necessary identifier (such as `<input name="book" />`) and, in fact, the data in any `input`, `textarea`, or `select` form field won't be sent at all if there isn't a `name` attribute present.

DISABLED AND READ-ONLY FIELDS

You can choose to disable an `input`, `textarea`, or `select` element with the `disabled` attribute (used in the format `disabled="disabled"`). When form fields are disabled in this fashion, the user won't be able to change them, and their values won't be sent when the form is submitted. This is a rarely used technique, but is sometimes used with JavaScript to disable/enable parts of a form depending on what options the user has selected elsewhere in the form.

Another attribute at your disposal is `readonly` (similarly used in the format `readonly="readonly"`), which is relevant to text type `input` elements and `textarea` elements. This simply won't let the user edit the text in the form field, so any initial value will remain. Like the `disabled` attribute, this is used only rarely, in conjunction with client-side scripts. The difference to the `disabled` attribute is that the value of an element set to `readonly="readonly"` *will* be sent when the form is submitted.

Putting Controls in Blocks

Before jumping in and adding form field elements to the form, keep in mind that the only valid direct content of a `form` element (when we're talking about the thinking person's choice, XHTML Strict—see Chapter 1, "Getting Started") are block-level elements and so `input`, `textarea`, and `select`, being inline, must sit inside one or more block-level element, such as a `div` (see Chapter 1) or a `fieldset` (see later in this chapter).

input

The `input` element is a gargantuan beast with many heads. With just this single element you can create text boxes, checkboxes, radio buttons, and more. You can specify which type of input control you want with the `type` attribute. For example, `type="text"` turns the element into a text box, `type="checkbox"` turns it into a

checkbox, and so on. There are 10 possible values for this attribute, and each type has its own peculiarities:

- text—for single-line text
- password—for obfuscated text
- checkbox—for a simple on or off
- radio—for selecting one of a number of options
- submit—for initiating the sending of the form data
- reset—for returning the form fields to their original values
- hidden—for data not seen, or edited by, the user
- image—for sending coordinates of where an image is clicked on
- file—for uploading files
- button—for shirts, pants, jackets, no… wait…

text

An `input` element with the attribute `type="text"` is a single-line text box—probably the most common form field, used for short pieces of textual information such as someone's name, email address, or credit card number. `text` is the default value for the `type` attribute (so you don't need to explicitly use the `type` attribute, if a text box is what you're after).

With `text` type `input` elements, you can also use the `maxlength` attribute, which limits the number of characters that can be typed into the text box. So the following example would create a text box into which the user can only type a maximum of four characters:

```
<input name="yearpublished" maxlength="4" />
```

The initial text contained within the text box can also be set with the `value` attribute. This is particularly handy either to give a pointer as to what the user should type in that text box (such as "Your name") or if values have been passed to the form, such as with "Remember me" functionality (whereby, with some clever scripting, a site can remember a user and fill in certain details automatically).

Create a Cork'd Account
Just fill out this form and we'll create your free account. Once you're done, you'll be able to start building wine lists, reviewing and rating wines, and finding drinking buddies.
Screen name:
PogoGogo
Your nickname here at Cork'd. One word.
Letters and numbers only. No spaces or special characters.
Email address:
pogogogo@htmldog.com
It's also your sign-in name, and has to be legit.
First name:
Pogo
What mom calls you.
Last name:
Gogo
What your army buddies call you.
Password:
Something you'll remember, but hard to guess.
Password confirmation:
Type it again. Think of it as a test.

FIGURE 9.2 Text input types and their closest relative, the password input.

password

The `password` type works like the `text` type, apart from one characteristic: As the user types, instead of the characters appearing in the text box, placeholder characters (usually asterisks or circular bullets, depending on the browser) will appear in their place. The idea behind this is that anyone peering over the user's shoulder won't be able to see what is being typed in.

```
<input type="password" name="pword" maxlength="20" />
```

www.htmldog.com/examples/inputtextboxes.html

checkbox

The `checkbox` type creates a simple checkbox, used to supply a yes/no, set/unset, or on/off option. By default, a browser will style this as a small empty square box, which, when selected, will display a "tick" inside the box.

Privacy:
Show my real name
If unchecked, people will only see your screen name.
Show my location
If unchecked, people will never see the city you live in.
Send me stuff
If checked, we may periodically send you Cork'd news, events or special announcements.

FIGURE 9.3 Checkboxes are used when more than one option can be selected.

You can also specify that the initial state of a checkbox should be selected ("checked" or "ticked") by adding the attribute and value combination `checked="checked"`. (In the past, simply `checked` would have been enough, but with XHTML, all attributes must have values.)

```
<input type="checkbox" name="modern" checked="checked" />
```

If a checkbox is not selected when the form data is submitted, no value will be sent. When the checkbox is selected, "on" will be sent as the value for the corresponding `name` unless the tag has a `value` attribute, in which case the value of that will be sent for the `name` instead.

www.htmldog.com/examples/inputcheckboxes.html

radio

Radio buttons, defined by the `radio` type, are similar to checkboxes, but the idea is that you can only select one option in a group at a time. You give a group of radio `input` elements the same name, and then when one of the radio buttons in that group is selected, any other radio buttons that were selected will be turned off.

```
<input type="radio" name="color" value="red" checked="checked" />
<input type="radio" name="color" value="orange" />
<input type="radio" name="color" value="blue" />
```

You really need to use the `value` attributes here. If you don't, the whole group will act the same as the checkbox—that is, if nothing is selected, nothing will be sent, but "on" (such as `color=on`) will be sent if any of the radio buttons in a group is selected, which isn't much help in discerning which of the options is selected. By supplying `value` attributes in each `input` element, the value of that attribute in the selected element will be sent, such as `color=orange`.

Add a New Wine to Cork'd
Tell the world about a new bottle that you've reviewed or keep track of a wine you own or want to buy.
FIRST, TELL US WHERE TO PUT THE WINE.
You've tried this wine (we'll add to your Wine Journal).
You own this wine (we'll add to your Wine Cellar).
You want to buy this wine (we'll add to your Shopping List).

FIGURE 9.4 Radio buttons are similar to checkboxes, but better when you want to allow only one option to be selected from a group.

As you can see from this example, once more you can also use `checked="checked"` to determine which radio button should initially be on.

`submit` and `reset`

There are other ways of submitting form data (namely with a bit of JavaScript), but the most common and easiest way is by hitting a submit button. The `submit` input type takes the form of a button and when this button is pressed, the form data will be sent to the value of the `action` attribute in the opening `form` tag.

By default, the text on the button will read something similar to "Submit Query" (depending on the browser), but this can be changed with the `value` attribute.

```
<input type="submit" value="Search" />
```

The `reset` input type creates a reset button, which, when clicked (or otherwise selected), will reset the form, with form fields returning to the values that they had when the page was first loaded.

Like submit, the text on the reset button ("Reset," by default) can be changed with the `value` attribute.

```
<input type="reset" value="Start again" />
```

With both submit and reset buttons, the `name` attribute isn't particularly necessary.

hidden

The `hidden` input type doesn't show up in the form and has nothing that the user can directly interact with. Sounds pretty useless on the face of it, doesn't it?

One use for hidden `input` elements is in passing data between form actions. If a user fills out one form and then needs to fill out a second form, a form-processing script can dynamically construct the second form and include in it data from the first form, or even data that it has calculated after the submission of the first form, such as a customer ID.

```
<input type="hidden" name="page" value="bobsbooks4.html" />
<input type="hidden" name="customerid" value="sk49fjp923j9fj9393" />
```

Another use is in setting variables for generic form-processing scripts. With some form-to-email scripts, for example, rather than expecting authors to mess about with the script itself (which can be daunting for those unfamiliar with the scripting language in question), they just need to specify things such as the email address that the form should be submitted to and the location of a "Thank you" page that the user will be directed to after submitting the form in the HTML via hidden `input` elements, such as:

```
<input type="hidden" name="recipient" value="whoever@wherever.com" />
<input type="hidden" name="thankyou" value="thankyou.html" />
```

image

The `image` type is like a cross between a `submit` type `input` element and an `img` element. Like an `img` element (See Chapter 4, "Images"), you can specify the file location of the image that will be used for the form field with the `src` attribute and a text alternative for the image with the `alt` attribute. Like the `submit` type, when the image is clicked on the form data will be sent.

On the rare occasions that `image` input types are used, they are most commonly used to provide graphical alternatives to a submit button. This degree of control over the appearance of the submit button may seem like a good thing, but it's not, really.

Firstly, it's purely presentational and takes away certain advantages of separating structure and presentation discussed throughout this book.

Secondly, as explained later in the "Presenting Forms" section, a standard submit button is pretty much instantly recognized by most users and messing with that familiarity makes the form more difficult to use.

Not only will the form be submitted when an image input element is selected, the pixel-coordinates where the user clicked on the image will also be sent. So, two values will be sent, such as:

```
image1.x=498
```

and

```
image1.y=128
```

and if the `value` attribute is used, a third value will be sent:

```
image1=valueofattribute
```

So image buttons can also serve as a server-side image map, whereby those coordinates can be processed and different actions can be taken depending on where the user clicked on the image. If the image was a map of the world, for example, a processing script could send users to a different page depending on which country they clicked.

Sounds nifty. Once again, it's not. Server-side image maps are rarely used, not only because of their specific nature and the complexity of the server-side programming required to fully exploit them, but because of their inaccessible nature: Not only do they rely on purely visual cues (such as country boundaries on a map), but they also rely on the user being able to click.

`file`

The `file` type allows users to select a file from their own computers, in order to upload that file. When the form is submitted, the selected file will be sent with the rest of the form data.

It should be remembered that when `type="file"` input elements are used, an additional `enctype` attribute must be added to the opening `form` tag with the value "multipart/form-data", so that when it is sent the server knows that it is getting more than textual data. The `method` attribute must also be set to `post`.

```
<form action="wherever.php" method="post" enctype="multipart/
form-data">
    <div>
        <input type="file" name="uploadfile" id="uploadfile" />
    </div>
</form>
```

www.htmldog.com/examples/inputfile.html

button

`button` input elements do absolutely nothing. Well, when it comes directly to form data, anyway. They are used to invoke client-side scripts (namely JavaScript—see Chapter 7, "Scripts & Objects") when the button is pressed. So whereas they play no part in submitted form data, they can be used to make other things in the form change, such as performing calculations and dynamically altering the value of a text box, for example.

textarea

A welcome break after the mad multitude of `input` element options, the `textarea` element is straightforward, having just one simple state. It works something like a big text-type `input` element, but is used for bigger chunks of textual data, usually with multiple lines, such as an address or a comment on a feedback form.

www.htmldog.com/examples/textarea.html

Unlike the `input` element, `textarea` has an opening and closing tag. Any text in between the opening and closing tag makes up the initial value of the element.

```
<textarea name="whatever" rows="10" cols="20">Type something here
</textarea>
```

In the above example, the text box will appear with "Type something here" inside the box.

Like using the value attribute in a `type="text"` input element, having initial text appear in this way can be useful in supplying extra information or instructions about

the kind of thing the user should type in the text area, and it can help with accessibility (see the "Accessible Forms" section later in this chapter). The disadvantage of doing this is that it requires more work on the users' part—selecting the text and deleting it before entering their own. For that reason, `textarea` is often used in the following way, with no content at all:

```
<textarea name="whatever" rows="10" cols="20"></textarea>
```

There is a peculiar XHTML anomaly that spoils the structure and presentation separation party. Inside the opening `textarea` tag, the attributes `rows` and `cols`, which determine the size of the text area, are not only valid but *required*. This will initially alter the width and height of the text area but you shouldn't be concerned by this since you can easily control the width and height with CSS.

select

`select` form fields present the user with a list (which is usually displayed as a drop-down menu), from which one or more options can be selected.

Key to their operation, another element is needed—the `option` element, which defines each option in the list.

```
<select name="book">
    <option>The Trial</option>
    <option>The Outsider</option>
    <option>Things Fall Apart</option>
    <option>Animal Farm</option>
</select>
```

www.htmldog.com/examples/select1.html

In cases such as the example above, when the user submits the form data, the value sent for the `select` element is the content of the selected `option` element (for example, if the third option was selected above, then "Things Fall Apart" would be sent as the value for "book"). You can supply different values for each `option` element by using the `value` attribute inside the opening `option` tag. When the `value` attribute is present, its value will be sent instead of the `option` element's content.

FIGURE 9.5 An alternative to checkboxes or radio buttons, select elements allow one or more selections from a list.

You can set one `option` element to be initially selected by using the `selected` attribute (in the form of `selected="selected"`).

In longer lists with obvious groupings, you can use `optgroup` elements, which supply a heading within the list (using the `label` attribute). Each option group can also be styled individually, so, if you choose, you can color some groups differently, for example.

```
<select name="book">
    <optgroup label="Camus">
        <option>The Outsider</option>
        <option>The Rebel</option>
        <option>The Plague</option>
    </optgroup>
    <optgroup label="Orwell">
        <option>Animal Farm</option>
        <option>Nineteen Eighty-Four</option>
        <option>Down and Out in Paris and London</option>
    </optgroup>
</select>
```

www.htmldog.com/examples/select2.html

By default, `select` elements will show one option at a time (and visually "drop down" the list of options when the element is clicked). You can choose to show more than one option at a time by setting the `size` attribute to the number of options you want. Instead of a drop-down list, browsers will display a sized select element as a fixed-height box containing the options, which, if all of the options do not fit in that box, will have a scrollbar to the right.

Also by default, the user can select only one option out of a `select` list. You can allow multiple selections by using the `multiple` attribute (in the form `multiple="multiple"`). When this is used, the user can select more than one option (usually by holding down a key, such as Ctrl, with every selection).

```
<select name="book" size="5" multiple="multiple">
<!-- etc -->
</select>
```

www.htmldog.com/examples/select3.html

Fieldsets

Imagine you have a long form with a multitude of form fields. Actually, it doesn't even need to be that long. Using fieldsets to group together common fields can help the user straight away by splitting up the form into chunks and making it more manageable. This can be done with the `fieldset` tag.

Fieldsets can additionally be given a caption/heading with a `legend` element, which must directly follow the opening `fieldset` tag.

```
<form action="whatever.php">
    <fieldset>
        <legend>Book Details</legend>
        <!-- lots of form fields -->    </fieldset>
    <fieldset>
        <legend>Some Other Details</legend>
        <!-- lots of form fields -->    </fieldset>
</form>
```

By default, a browser will render a fieldset with a border around it and a legend as a heading breaking the top border. You can choose to turn off the border (`border: 0`), but you won't have much success in styling it any other way—a legend will always insist on sitting on top of a fieldset border.

Note that all of the bare-bone examples mentioned so far in this chapter contain fieldsets and legends.

Accessible Forms

The first step towards accessible forms is to have a sensible design: well spaced-out form fields with labels that are clearly associated with them are going to make things much easier to use for anyone—and especially someone with any level of visual impairment.

Grouping items with elements such as `optgroup` and `fieldset` will also help in splitting up the form and visualizing distinct areas as well as aiding assistive technology.

There are also steps that can be taken that are similar to the accessibility issues regarding links. Using `tabindex` and `accesskey` attributes in the `input`, `textarea`, and `select` tags, to aid navigation for those who do not use pointing devices such as a mouse, will achieve the same benefits and drawbacks as discussed in Chapter 3, "Links".

As with any element on a page, `title` attributes can also be used to provide more information, in this case possibly to explain with greater detail than a label what the user should enter in a field.

Labels

Every form field element should be accompanied by a `label` element. It's not particularly difficult; in fact, every form field should have a textual label explaining what a form field is for anyway—the label element just formalizes the matter. A `label` element links a label with a form field element, providing an explicit link between the two rather than relying on visual proximity or adjacency within the HTML code.

The value of the `for` attribute associates the label with the form field whose `id` has a corresponding value, such as:

```
<label for="rasputin">Rasputin:</label>
<input name="whatever" id="rasputin" />
```

No, this isn't a particularly practical example. In most cases, you will probably find the form field having the same values for both the name and id attributes:

```
<label for="yourname">Your name:</label>
<input name="yourname" id="yourname" />
<label for="youraddress">Your address:</label>
<textarea name="youraddress" id="youraddress" />
```

The added benefit of `label` elements is one of usability, particularly with checkboxes and radio buttons. When the label is clicked, the focus will be placed on the associated form field element. In the case of checkboxes and radio buttons, this means that not only can you check or uncheck the element via the small area of the element itself, you can also do so by clicking on the label, providing a much larger (and easier) area to click.

The web pages behind the figures on this page all employ labels. Look at *www .htmldog.com/examples/inputcheckboxes.html,* for example, to see them at work.

Styling Form Fields

There's something slightly special about form fields—a browser will actually pull in a widget that is part of the operating system to make a form field element. The implications of this are that it's nigh on impossible to achieve a uniform style across all browsers (and different OSs) and you are limited as to the extent to which you can style certain aspects of these form fields.

Because form items come not only from the browser but also from the operating system, web pages aren't the only place that computer users come across radio buttons and text boxes, etc. They are a familiar element of OS settings and of software programs that run on them. The borders of form fields are purposefully made to make the element look three dimensional—text boxes, for example, appearing lowered and buttons appearing raised, to make them "feel" more tactile—suggesting a user can interact with them.

And for a similar reason that it is a common suggestion to keep text links blue and underlined (see Chapter 3) due to users' familiarity with what that style signifies, it is also a common suggestion to leave forms in their default style.

At the possible compromise of usability, many do opt to alter the style of form fields, but there are some limitations.

One example is buttons. Browsers such as Opera and Safari have their own style of buttons. These browsers actually go as far as ignoring any decoration, such as borders or colors, that you attempt to apply to them (whereas other browsers, such as Internet Explorer, will give you free rein).

There are some safe changes that can be made to many form fields. You'll probably want control over dimensions, rather than relying on the default rendering of those elements. As with any other box, you can simply use the properties `height` (particularly useful for text areas) and `width`. Changing dimensions is absolutely fine and pretty much hassle-free because users are quite used to seeing form fields of various sizes.

There are a few other properties you can play with when it comes to forms. There are no CSS properties specific to forms, but you can apply colors, padding, borders, and margins to most form elements, just like any other visible element. As you will see, however, it's not all smooth sailing.

Borders

A quick, easy, and common way to radically change the appearance of form fields such as text boxes is to take control of their borders, using the `border` property (see Chapter 5, "Layout").

```
input, textarea { border: 1px solid #ccc }
```

Some might argue that this thin border is more visually appealing, but it should be kept in mind that this may be at the detriment of usability. A compromise might be something like this:

```
input, textarea {
    border: 1px solid;
    border-color: #666 #ccc #ccc #666;
}
```

This will apply a 1-px border, overriding the thicker default border, but will keep the three-dimensional effect users are used to, by applying different shades to the top/left and bottom/right borders.

When it comes to select menus, you're stuffed as far as customized borders go. Most browsers won't apply CSS borders to select elements at all. So if your form includes these stubborn little blighters, you're probably better off leaving the borders of all of the form field elements well alone, and settling for the default, for the sake of consistency.

A similar problem arises when it comes to checkboxes and radio buttons. The gray borders that make up the actual square or circle are also stuck in their ways and refuse to change.

Fonts

You can specify the font details of text that will appear in text boxes and text areas just as you can for text in other elements on the web page. The `input`, `textarea`, and `select` elements will not inherit any font properties, however, and they all have different initial properties by default—`textarea` has a Courier font and `input` has a sans-serif font, for example. To set things such as `color`, `font-family`, and `font-size`, then, you need to be explicit.

```
input, textarea { font: 1em arial, Helvetica, sans-serif }
```

Backgrounds

As with borders, it is questionable whether the backgrounds of form fields and buttons should be anything other than the default—white for text boxes and text areas and gray for buttons. You do have the option of specifying either background colors or background images for your controls, although many browsers will ignore any background settings for checkboxes and radio buttons, leaving them white.

A clever technique that could quite possibly aid usability is to change the background color of form fields such as text boxes and text areas when they are in focus, making that field stand out from the others and make it clearer where the user is on the page. This can be achieved with the dynamic pseudo-class `:focus`.

```
input:focus, textarea:focus { background: #eee }
```

This could, of course, be used to change any property of a form field when it has focus, such as the border.

As explained in Chapter 7, Internet Explorer 6 (and earlier) doesn't support this, but can this can easily be fixed with a little Suckerfish JavaScript. See the article at *www.htmldog.com/articles/suckerfish/focus* or just see it in action at *www.htmldog.com/articles/suckerfish/focus/example*.

chapter 10

Multiple Media

CONJURE AN IMAGE in your mind of someone looking at a web page. Go on, just do it. Humor me.

Chances are you will be thinking of someone sitting at a desk with a desktop or laptop computer displaying the web page in all its lavish glory on a 15- to 30-inch color screen. In all likelihood, the majority of people looking at your web pages *will* be in this situation, but there will also be those who can't use a screen at all, an increasing number of people who are using all sorts of mobile devices to access the web, and even just those who simply want to print out a web page.

Web pages can be accessed and consumed in many ways.

The good news is that if you've been building your pages following web standards then you'll already be accommodating multiple media and devices to a much greater degree than if you hadn't. And with a few little tricks you can further optimize for different devices.

The purpose of this chapter is to look outside the usual web surfing as a computer-and-monitor-on-a-desk experience and explore web pages being consumed in different ways—namely through screen-readers, mobile devices, and, in particular, print. It is an epilogue, if you will, demonstrating one of the great benefits of best-practice web standard XHTML and CSS.

Screen-Readers

This book has touched on the subject of accessibility a number of times, including the accommodation of assistive devices, such as screen-readers, that web users with physical impairments might use.

A screen-reader aids those who cannot see a monitor adequately by reading out the content and other elements that might otherwise appear on a computer screen, including web pages.

How do we accommodate screen-readers? Well, we should already be doing it by using semantic HTML.

If the content in a web page is arranged in a sensible, logical order (rather than spread all over the place in a multitude of table cells, for example), and paragraphs are found in `p`s, unordered lists are found in `ul`s, and tables really are tables, then it's all gravy, baby.

These pieces of software are immense, clever things that attempt to decipher the crappiest of code. With currently supported technologies, there is nothing we can do to directly target screen-readers, but just by coding in the clearest, proper manner, our web pages are more likely to be understood as they are meant to be.

Mobile Devices

OK, so mobile devices do have monitors, but they're itsy-bitsy ones, hardly worthy of the mighty mantle of a true monitor.

I shouldn't need to spell out the major differences in mobile screens and your standard hulk of a desktop monitor, but I will: S.M.A.L.L. S.C.R.E.E.N. Apart from the screen-size difference, there are also issues with fiddly input (there's no full-on QWERTY keyboard here), a greater potential for slower access times than with standard Internet connections, and a whole plethora of browser compatibility differences between devices.

So we can, in theory, simply restyle our existing content so that it's more suitable for the mobile Web. There are two opposing camps, though, who sing and dance about it in a faux-aggressive manner just like the gangs in Michael Jackson videos

from the 1980s. One camp, weighed down by superfluous buckles, dances for the honor of restyled universal, multimedia content which, you know, is a lovely theory, but the other camp, brandishing plastic knives while balancing on roller skates, sings "You really need to re-purpose your content to better suit the medium."

Under most circumstances (those websites with anything other than the most minimal content), I think I'd side with the latter camp (although I'd object if they asked me to wear roller skates). Although the universality side of web standards is just peachy, reams of content that might be fine on a single standard web page usually need to be broken down into more bite-sized chunks (or even replaced with more concentrated content) for smaller-screen devices on which information is consumed so differently. Having said that, most of us aren't in the position, or even inclined, to make the effort to produce separate mobile-friendly content. But we can at least take the easy step of making things look a little tidier on small-screen devices.

We will look at *how* to do this later in the chapter, but as for what we might do, we're going to want to think vertically and forget about multiple columns, tone down heavy, detailed presentational imagery. It might even be a good idea to hide nonessential components (such as repeated navigation).

It's worth keeping in mind that, particularly when it comes to CSS, if you think you've got problems with cross-browser compatibility between Firefox and IE, it's a whole new league of diversity in the mobile world, with a multitude of different devices, different operating systems, and different browsers. What looks like Marilyn Monroe on one phone (with good support) might look like Marilyn Manson on another (with poor support). So wield your power wisely—consider basic styling, or maybe even simply relying on the browser style sheet. If your HTML is good (which, now you're on the final chapter, it can't fail to be, surely), then your web pages should work better on mobile devices than most other websites out there.

Print

So we've gathered that good HTML is important for accommodating different devices and media. That's grand. CSS hasn't quite had a look-in as yet, thanks to the aural nature of screen-readers, and the compatibility issues with mobile devices.

When it comes to printing out web pages, there are a number of style changes we can make to increase suitability to the medium:

- **Font type**: While sans-serif fonts such as Arial, Helvetica, or Verdana are easier to read onscreen, serif fonts, such as Times New Roman, are easier to read in print.
- **Font size**: Unlike screen, print is an absolute medium, so we might as well go with a traditionally print-related unit: points.
- **Useless components**: The likes of navigation bars and forms—anything that requires user interaction on a functional web page—are pretty useless when printed out, so there's no use printing them out at all. This is another reason why "click here" and its kin are so loathsome: When such phrases are printed out, no matter how hard you try to click, unless it's magic paper and you're using Harry Potter's mouse, very little is going to happen.
- **Page width**: Ensuring that the content you're printing is liquid, you can ensure that it will make the best use of space on a page (if it was originally a thin column on screen, for example), and also make sure it will all fit on one page (if it was a wide column, which might be too wide for a piece of paper). There are many arguments here about fixed versus liquid layouts—we're all used to standard paper sizes (which, it might be worth noting if you were considering messing with fixed widths, are different in different countries around the world).
- **Colors**: Detailed and colorful as a flock of macaws though your web page might be, be prepared to think "magpie" when it comes to print. Black and white printers are still common (as is the choice to print in black and white), so it wouldn't be a good idea to rely on color. It's worth remembering, though, that white-on-black can also be ink-guzzling and messy.
- **Background images**: By default, most browsers will not print out background images (and it takes some digging to switch the option on), so, like colors, don't rely on them.

A Sample Print Stylesheet

To take these different approaches into account, a print-specific CSS might look something like this:

```
body {
    font: 12pt "Times New Roman", Times, serif;
    color: black;
    background: white;
}
#navigation, form {
    display: none;
}
a {
    color: black;
    text-decoration: none;
}
#content {
    margin: 0;
    width: 100%;
}
```

This essentially covers the list of points made previously. The `body` rule sets the font to a sensible size in points, and to a serif font. The navigation area and all forms are completely pulled out of the picture. Links are made to look like surrounding text (because there's nothing particularly special about them when printed out). Assuming we were putting this on top of a fixed layout, the content area is made to fit the full width, and the margin, which might have been used to accommodate the navigation area (see Chapter 5, "Layout"), is zeroed.

Applying Media-Specific CSS

Does accommodating all of these media-specific styles mean we have to serve up two versions of the same website? Not at all (although, as mentioned, we might choose to when it comes to pages for mobile devices, for example). What we can do is target certain styles for certain media, leaving the HTML well alone (it should simply be, after all, meaningful content, and presentation-free).

"PRINTER FRIENDLY" ALREADY, THANKS

"Click here for a printer-friendly version" is not an uncommon phrase found on the Web. But you don't really want to construct two versions of every page, and you don't have to. Thanks to the separation of content and presentation, your existing web pages can be printer friendly as they are.

The `media` Attribute

If you remember from Chapter 1, "Getting Started," to apply CSS to our HTML we can use either the `link` element:

```
<link rel="stylesheet" type="text/css" href="core.css" />
```

Or we can use the `style` element / `@import` rule combo:

```
<style type="text/css">@import url("core.css");</style>
```

To cut to the chase, we can simply use the `media` attribute to apply a style sheet to our HTML when it's being read by a particular device or intended for a particular medium. So, if we want one style sheet for the standard desktop scenario, and one for when the web pages are printed out, we could do:

```
<link rel="stylesheet" type="text/css" media="screen" href="screen.
css" />
<link rel="stylesheet" type="text/css" media="print" href="print.
css" />
```

Or:

```
<style type="text/css" media="screen">@import url("screen.css");
</style>
<style type="text/css" media="print">@import url("print.css");
</style>
```

And it's as simple as that.

Note that the values for the `media` attribute used here are `screen` and `print`, but you can also use `handheld` (for mobile devices, although not all of them recognize this, hence one of the compatibility problems). Other options, which aren't widely supported, are `projection`, `braille`, and `speech`.

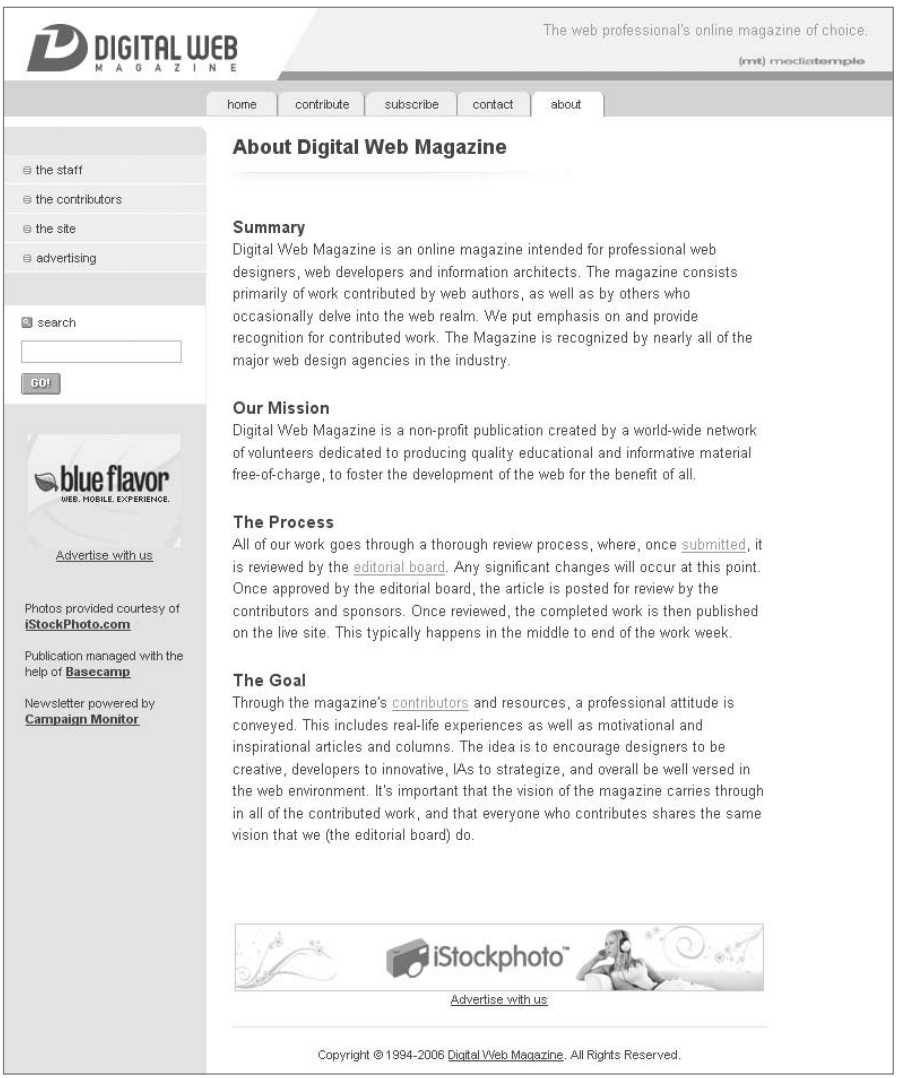
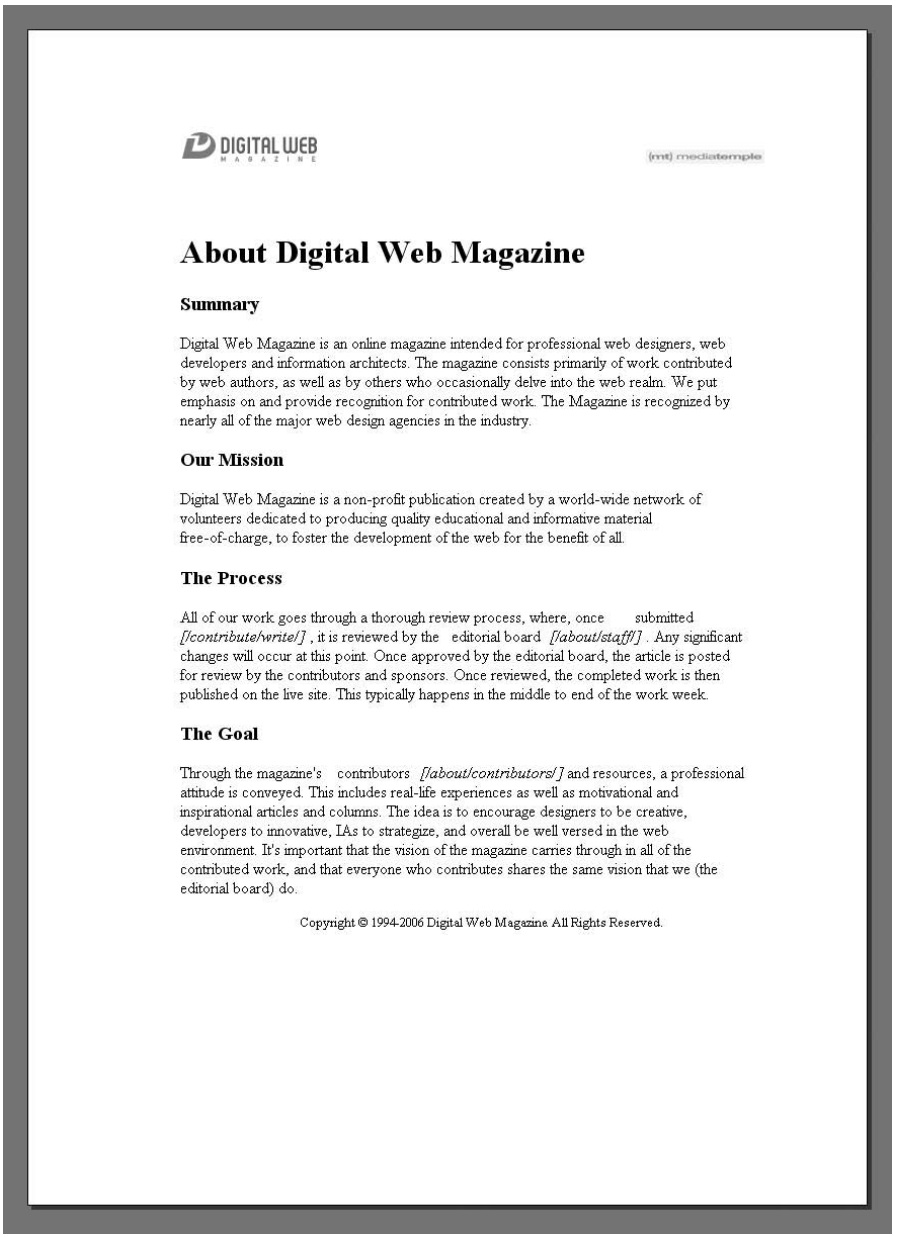

DIGITAL WEB MAGAZINE
The web professional's online magazine of choice.
(mt) mediatemple
home
contribute
subscribe
contact
about
the staff
the contributors
the site
advertising
search
GO!
blue flavor
WEB. MOBILE. EXPERIENCE.
Advertise with us
Photos provided courtesy of iStockPhoto.com
Publication managed with the help of Basecamp
Newsletter powered by Campaign Monitor
About Digital Web Magazine
Summary
Digital Web Magazine is an online magazine intended for professional web designers, web developers and information architects. The magazine consists primarily of work contributed by web authors, as well as by others who occasionally delve into the web realm. We put emphasis on and provide recognition for contributed work. The Magazine is recognized by nearly all of the major web design agencies in the industry.
Our Mission
Digital Web Magazine is a non-profit publication created by a world-wide network of volunteers dedicated to producing quality educational and informative material free-of-charge, to foster the development of the web for the benefit of all.
The Process
All of our work goes through a thorough review process, where, once submitted, it is reviewed by the editorial board. Any significant changes will occur at this point. Once approved by the editorial board, the article is posted for review by the contributors and sponsors. Once reviewed, the completed work is then published on the live site. This typically happens in the middle to end of the work week.
The Goal
Through the magazine's contributors and resources, a professional attitude is conveyed. This includes real-life experiences as well as motivational and inspirational articles and columns. The idea is to encourage designers to be creative, developers to innovative, IAs to strategize, and overall be well versed in the web environment. It's important that the vision of the magazine carries through in all of the contributed work, and that everyone who contributes shares the same vision that we (the editorial board) do.
iStockphoto
Advertise with us
Copyright © 1994-2006 Digital Web Magazine. All Rights Reserved.

FIGURE 10.1 *www.digital-web.com* on screen…



DIGITAL WEB MAGAZINE

(mt) mediatemple

About Digital Web Magazine

Summary

Digital Web Magazine is an online magazine intended for professional web designers, web developers and information architects. The magazine consists primarily of work contributed by web authors, as well as by others who occasionally delve into the web realm. We put emphasis on and provide recognition for contributed work. The Magazine is recognized by nearly all of the major web design agencies in the industry.

Our Mission

Digital Web Magazine is a non-profit publication created by a world-wide network of volunteers dedicated to producing quality educational and informative material free-of-charge, to foster the development of the web for the benefit of all.

The Process

All of our work goes through a thorough review process, where, once submitted *[/contribute/write/]* , it is reviewed by the editorial board *[/about/staff/]* . Any significant changes will occur at this point. Once approved by the editorial board, the article is posted for review by the contributors and sponsors. Once reviewed, the completed work is then published on the live site. This typically happens in the middle to end of the work week.

The Goal

Through the magazine's contributors *[/about/contributors/]* and resources, a professional attitude is conveyed. This includes real-life experiences as well as motivational and inspirational articles and columns. The idea is to encourage designers to be creative, developers to innovative, IAs to strategize, and overall be well versed in the web environment. It's important that the vision of the magazine carries through in all of the contributed work, and that everyone who contributes shares the same vision that we (the editorial board) do.

Copyright © 1994-2006 Digital Web Magazine. All Rights Reserved.

FIGURE 10.2 ...*www.digital-web.com* in print,

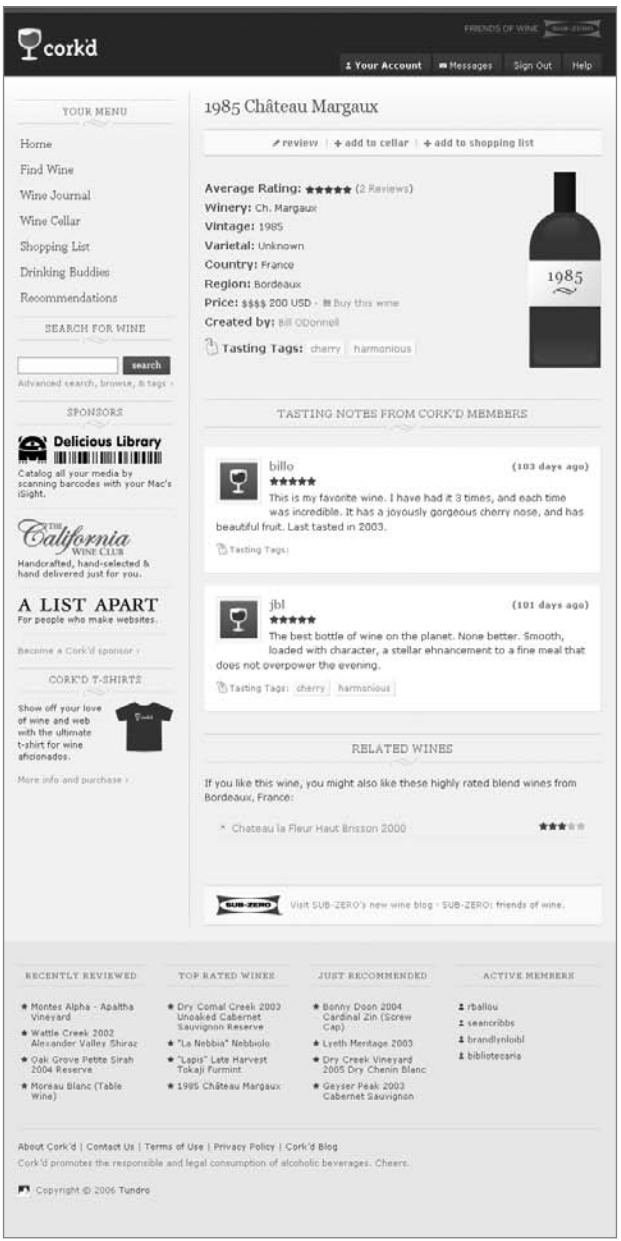

corkd
FRIENDS OF WINE
Your Account
Messages
Sign Out
Help
YOUR MENU
Home
Find Wine
Wine Journal
Wine Cellar
Shopping List
Drinking Buddies
Recommendations
SEARCH FOR WINE
search
Advanced search, browse, & tags ›
SPONSORS
Delicious Library
Catalog all your media by scanning barcodes with your Mac's iSight.
The California Wine Club
Handcrafted, hand-selected & hand delivered just for you.
A LIST APART
For people who make websites.
Become a Cork'd sponsor ›
CORK'D T-SHIRTS
Show off your love of wine and web with the ultimate t-shirt for wine aficionados.
More info and purchase ›
1985 Château Margaux
review | + add to cellar | + add to shopping list
Average Rating: ★★★★★ (2 Reviews)
Winery: Ch. Margaux
Vintage: 1985
Varietal: Unknown
Country: France
Region: Bordeaux
Price: $$$$ 200 USD · Buy this wine
Created by: Bill ODonnell
Tasting Tags: cherry harmonious
1985
TASTING NOTES FROM CORK'D MEMBERS
billo (103 days ago)
★★★★★
This is my favorite wine. I have had it 3 times, and each time was incredible. It has a joyously gorgeous cherry nose, and has beautiful fruit. Last tasted in 2003.
Tasting Tags:
jbl (101 days ago)
★★★★★
The best bottle of wine on the planet. None better. Smooth, loaded with character, a stellar ehnancement to a fine meal that does not overpower the evening.
Tasting Tags: cherry harmonious
RELATED WINES
If you like this wine, you might also like these highly rated blend wines from Bordeaux, France:
Chateau la Fleur Haut Brisson 2000 ★★★
SUB-ZERO
Visit SUB-ZERO's new wine blog - SUB-ZERO: friends of wine.
RECENTLY REVIEWED
Montes Alpha - Apaltha Vineyard
Wattle Creek 2002 Alexander Valley Shiraz
Oak Grove Petite Sirah 2004 Reserve
Moreau Blanc (Table Wine)
TOP RATED WINES
Dry Comal Creek 2003 Unoaked Cabernet Sauvignon Reserve
"La Nebbia" Nebbiolo
"Lapis" Late Harvest Tokaji Furmint
1985 Château Margaux
JUST RECOMMENDED
Bonny Doon 2004 Cardinal Zin (Screw Cap)
Lyeth Meritage 2003
Dry Creek Vineyard 2005 Dry Chenin Blanc
Geyser Peak 2003 Cabernet Sauvignon
ACTIVE MEMBERS
rballou
seancribbs
brandlynloibl
bibliotecaris
About Cork'd | Contact Us | Terms of Use | Privacy Policy | Cork'd Blog
Cork'd promotes the responsible and legal consumption of alcoholic beverages. Cheers.
Copyright © 2006 Tundra

FIGURE 10.3 *corkd.com* on screen…

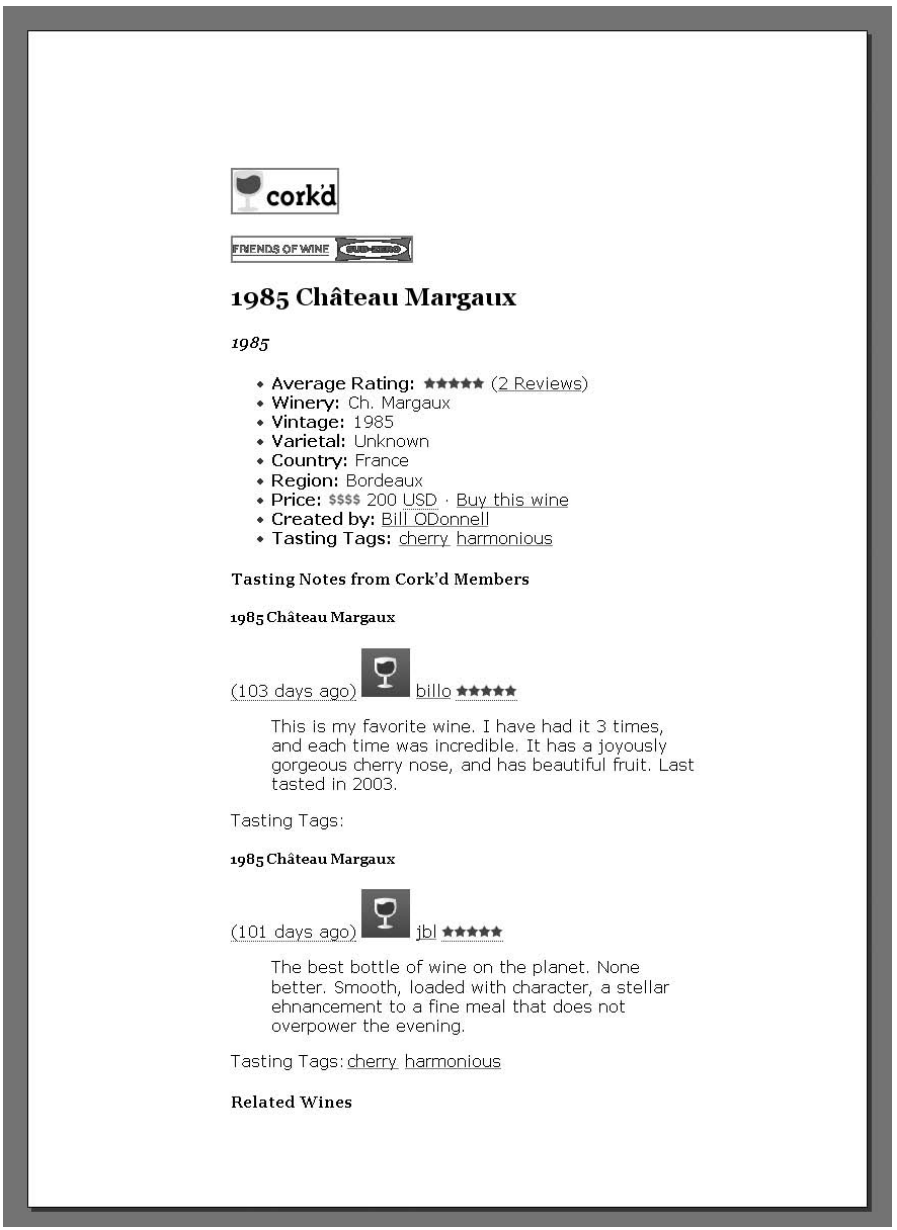

corkd
FRIENDS OF WINE
1985 Château Margaux
1985
Average Rating: ★★★★★ (2 Reviews)
Winery: Ch. Margaux
Vintage: 1985
Varietal: Unknown
Country: France
Region: Bordeaux
Price: $$$$ 200 USD · Buy this wine
Created by: Bill ODonnell
Tasting Tags: cherry harmonious
Tasting Notes from Cork'd Members
1985 Château Margaux
(103 days ago) billo ★★★★★
This is my favorite wine. I have had it 3 times, and each time was incredible. It has a joyously gorgeous cherry nose, and has beautiful fruit. Last tasted in 2003.
Tasting Tags:
1985 Château Margaux
(101 days ago) jbl ★★★★★
The best bottle of wine on the planet. None better. Smooth, loaded with character, a stellar ehnancement to a fine meal that does not overpower the evening.
Tasting Tags: cherry harmonious
Related Wines

FIGURE 10.4A *...corkd.com* in print.

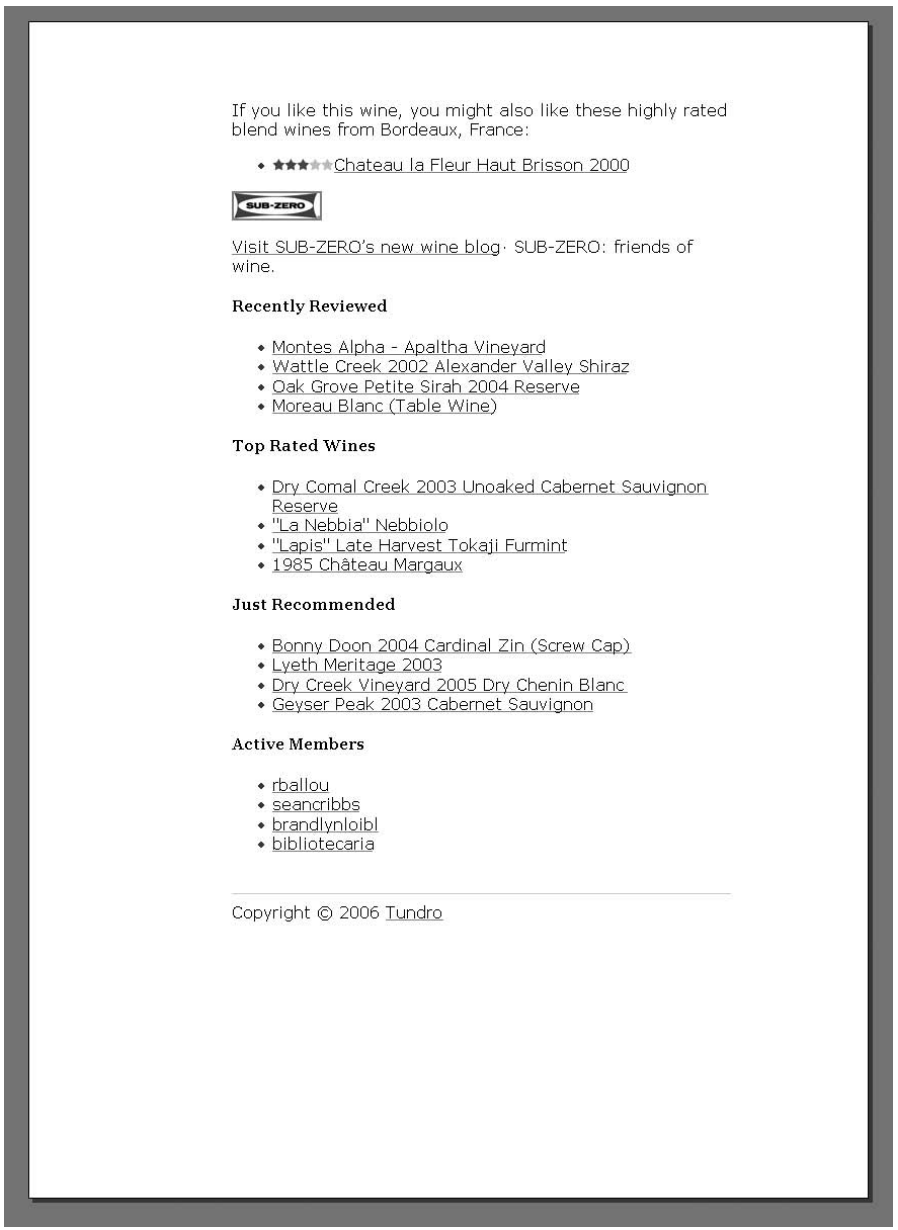

If you like this wine, you might also like these highly rated blend wines from Bordeaux, France:
★★★☆☆Chateau la Fleur Haut Brisson 2000
SUB-ZERO
Visit SUB-ZERO's new wine blog · SUB-ZERO: friends of wine.
Recently Reviewed
Montes Alpha - Apaltha Vineyard
Wattle Creek 2002 Alexander Valley Shiraz
Oak Grove Petite Sirah 2004 Reserve
Moreau Blanc (Table Wine)
Top Rated Wines
Dry Comal Creek 2003 Unoaked Cabernet Sauvignon Reserve
"La Nebbia" Nebbiolo
"Lapis" Late Harvest Tokaji Furmint
1985 Château Margaux
Just Recommended
Bonny Doon 2004 Cardinal Zin (Screw Cap)
Lyeth Meritage 2003
Dry Creek Vineyard 2005 Dry Chenin Blanc
Geyser Peak 2003 Cabernet Sauvignon
Active Members
rballou
seancribbs
brandlynloibl
bibliotecaria
Copyright © 2006 Tundro

FIGURE 10.4B ...*corkd.com* in print.

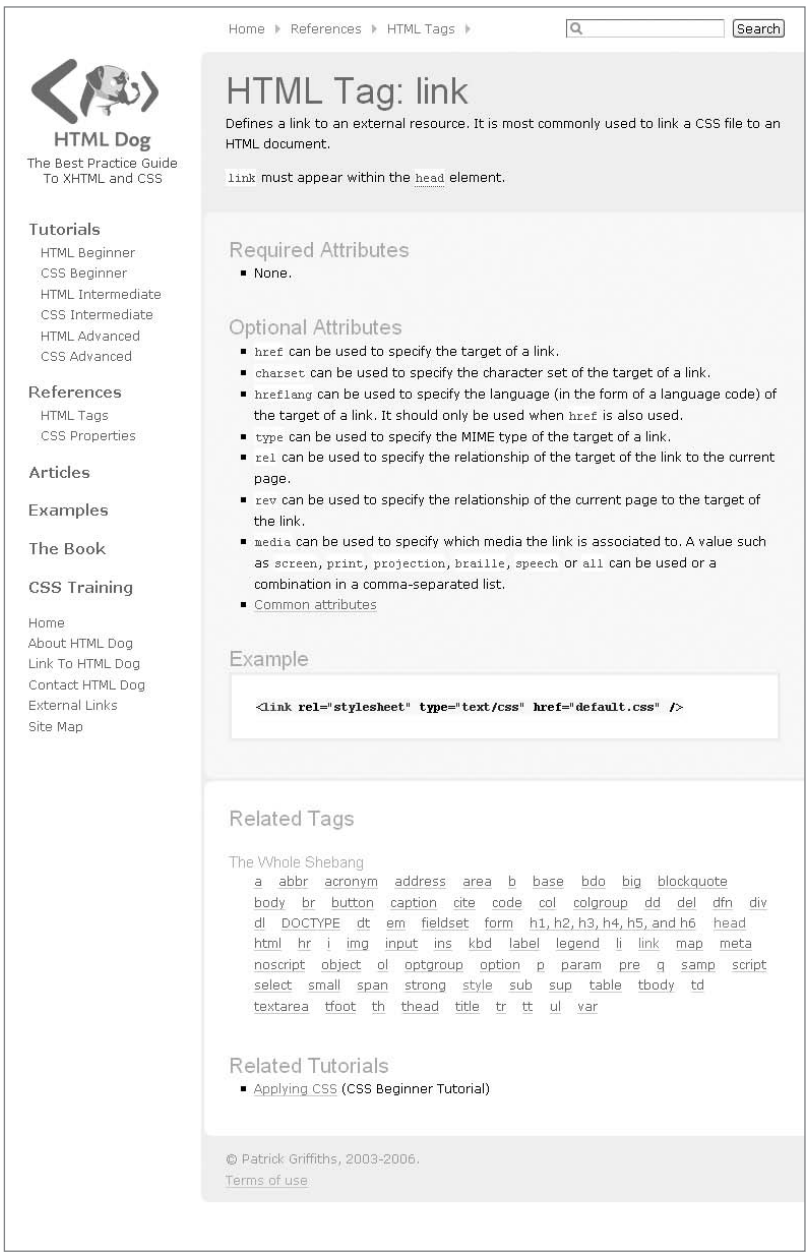

Home ▸ References ▸ HTML Tags ▸
Search
HTML Dog
The Best Practice Guide To XHTML and CSS
Tutorials
HTML Beginner
CSS Beginner
HTML Intermediate
CSS Intermediate
HTML Advanced
CSS Advanced
References
HTML Tags
CSS Properties
Articles
Examples
The Book
CSS Training
Home
About HTML Dog
Link To HTML Dog
Contact HTML Dog
External Links
Site Map
HTML Tag: link
Defines a link to an external resource. It is most commonly used to link a CSS file to an HTML document.
link must appear within the head element.
Required Attributes
None.
Optional Attributes
href can be used to specify the target of a link.
charset can be used to specify the character set of the target of a link.
hreflang can be used to specify the language (in the form of a language code) of the target of a link. It should only be used when href is also used.
type can be used to specify the MIME type of the target of a link.
rel can be used to specify the relationship of the target of the link to the current page.
rev can be used to specify the relationship of the current page to the target of the link.
media can be used to specify which media the link is associated to. A value such as screen, print, projection, braille, speech or all can be used or a combination in a comma-separated list.
Common attributes
Example
<link rel="stylesheet" type="text/css" href="default.css" />
Related Tags
The Whole Shebang
a abbr acronym address area b base bdo big blockquote body br button caption cite code col colgroup dd del dfn div dl DOCTYPE dt em fieldset form h1, h2, h3, h4, h5, and h6 head html hr i img input ins kbd label legend li link map meta noscript object ol optgroup option p param pre q samp script select small span strong style sub sup table tbody td textarea tfoot th thead title tr tt ul var
Related Tutorials
Applying CSS (CSS Beginner Tutorial)
© Patrick Griffiths, 2003-2006.
Terms of use

FIGURE 10.5 *www.htmldog.com* on screen…

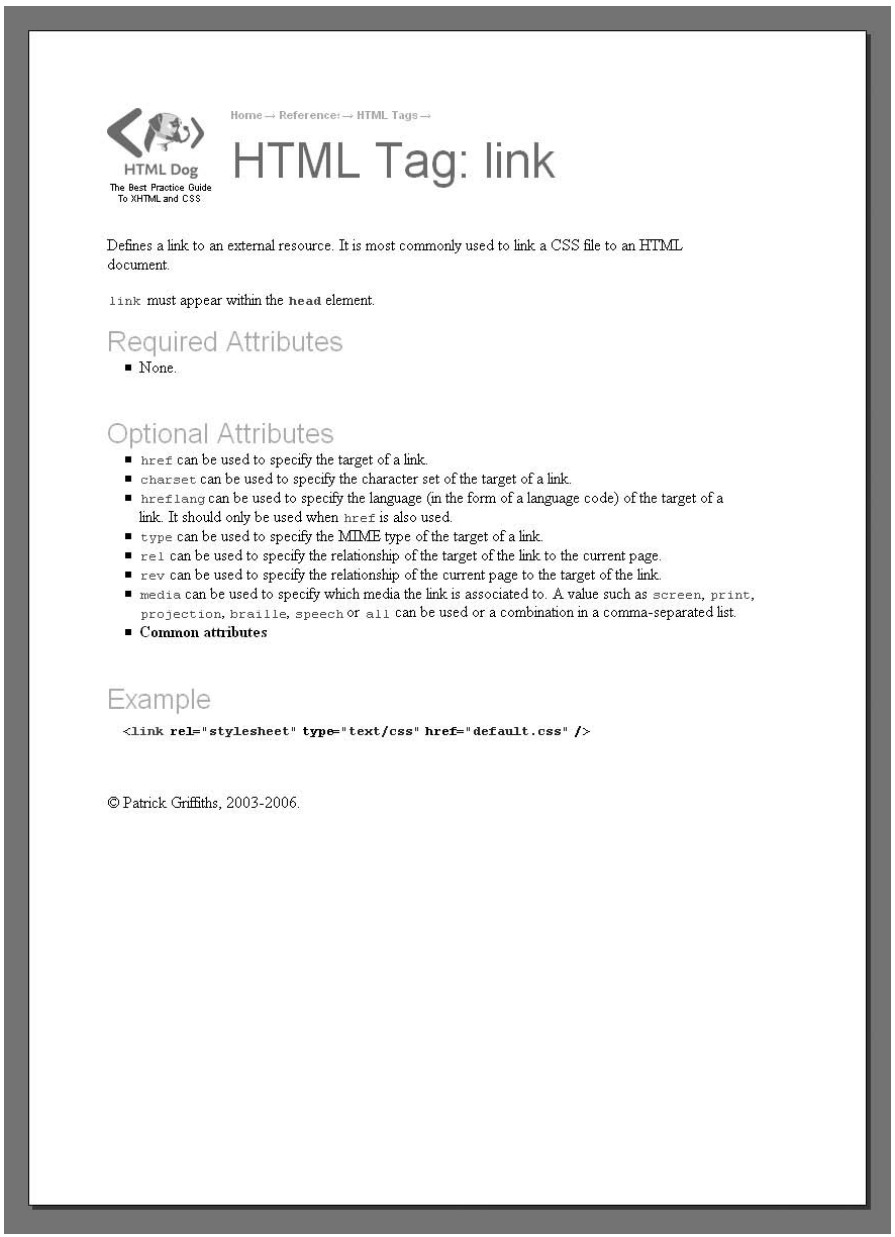

HTML Dog
The Best Practice Guide To XHTML and CSS
Home → Reference: → HTML Tags →
HTML Tag: link
Defines a link to an external resource. It is most commonly used to link a CSS file to an HTML document.
link must appear within the head element.
Required Attributes
None.
Optional Attributes
href can be used to specify the target of a link.
charset can be used to specify the character set of the target of a link.
hreflang can be used to specify the language (in the form of a language code) of the target of a link. It should only be used when href is also used.
type can be used to specify the MIME type of the target of a link.
rel can be used to specify the relationship of the target of the link to the current page.
rev can be used to specify the relationship of the current page to the target of the link.
media can be used to specify which media the link is associated to. A value such as screen, print, projection, braille, speech or all can be used or a combination in a comma-separated list.
Common attributes
Example
<link rel="stylesheet" type="text/css" href="default.css" />
© Patrick Griffiths, 2003-2006.

FIGURE 10.6 ...*www.htmldog.com* in print.

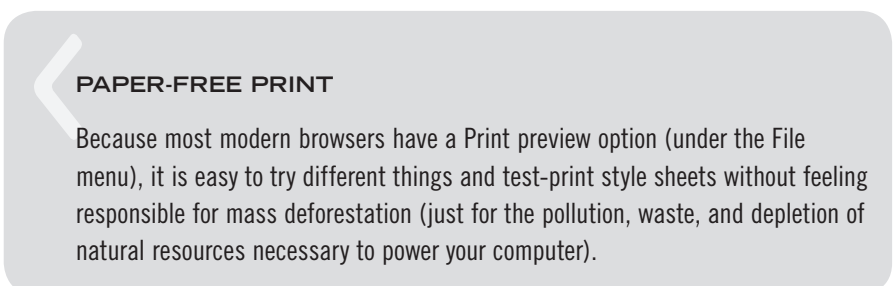
Separate or Cascading

There are two approaches to consider when applying multiple style sheets. You can have an individual, stand-alone style sheet for each situation (such as one for screen and one for print), or you can have a central, core style sheet, on top of which another style sheet, for a certain situation (such as one for everything, and one only for print). The former is a separated approach; the latter is a cascading approach. While the examples above show a separated approach, a cascading approach would look something like this:

```
<link rel="stylesheet" type="text/css" href="screen.css" />
<link rel="stylesheet" type="text/css" media="print" href="print.
css" />
```

Because the first `link` element doesn't have a `media` attribute, it will apply that CSS file to everything. When it comes to print, however, the second `link` element will additionally add the *print.css* file.

Each approach has advantages and disadvantages. Separate may require more original rules (such as setting colors or font sizes), whereas cascading might require more overriding rules (such as making borders or backgrounds invisible).



PAPER-FREE PRINT

Because most modern browsers have a Print preview option (under the File menu), it is easy to try different things and test-print style sheets without feeling responsible for mass deforestation (just for the pollution, waste, and depletion of natural resources necessary to power your computer).

@media

Oh, and before I go I should mention `@media`.

`@media` isn't only the name of the greatest web design conference this side of Wonderland, it's a special CSS selector that enables you to plonk media-specific

styles directly into an existing style sheet, effectively creating a cascading approach, but within one file, and placing the media selection in the hands of CSS rather than via the media attribute in HTML:

```
@media print {
    body {
        font: 12pt "Times New Roman", Times, serif;
    }
    #navigation {
        display: none;
    }
    /* etc. */
}
```

The media type following "`@media`" can be a single media type, or a comma-separated list. So you could have:

```
@media print, handheld { /* specific rules here */ }
```

for example.

In Conclusion

Well, kids, there you have it: Best-practice (X)HTML and CSS in one handy volume.

The next step? Practice. Experiment. Play around. Read more web design and development books and weblogs for inspiration. HTML Dog has explained how to use your tools, but there's no substitute for working with them and getting a feel for how they work in practice. The more you use them, the better a web craftsman you will become.

This page intentionally left blank

appendix

XHTML Reference

THIS HTML REFERENCE covers the common attributes and tags of XHTML 1.0 Strict.

If it's not here then there's a good reason for not using it. A brief overview of some of the more common "Bad Tags & Rotten Attributes" can be found at the end of this appendix.

More details of tag and attribute usage can be found in the indicated chapters and the syntax and application of HTML can be found in Chapter 1, "Getting Started."

Tags

The possible attributes that can be used in a tag will take one of these formats (see Chapter 1, "Getting Started," for more):

- attributename—An optional attribute, such as `href` in `<a href="alaska.html">`.
- attributename (required)—An attribute that must be used.
- [Core attributes]— A collection of general attributes that can be applied to most tags—`class`, `id`, `title`, and `style`.
    - **`class`**—Used to reference elements. More than one element can have the same class name. Can be used by CSS to target elements.
    - **`id`**—Identifies a unique element; that is, only one element on a page can have any given (case sensitive) id attribute value. Can be used by CSS to target elements.

    - **title**—Adds a title to an element.
    - **style**—Used to apply inline styles. Should be avoided if possible.
- [I18n attributes]—Internationlization—dir, lang, and xml:lang.
    - **dir**—The direction of content. Values can be **ltr** (left to right, for languages such as English) or **rtl** (right to left, for languages such as Arabic).
    - **lang** and **xml:lang**—The language of the content of an element, such as **en** for English, or **de** for German.
- [Event attributes]—onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup. You can read more about event attributes, and why you should try to avoid using them, in Chapter 7, "Scripts & Objects."
- [Common attributes]—The core, I18n, and event attributes combined.

Content describes the valid content of an element—the content that can appear between the opening and closing tags. For the purposes of this reference, the following terms apply to content:

- Empty—The tag closes itself (such as <tag />).
- None—Doesn't contain anything (such as <tag></tag>).
- Text—Processed character data (which is text and processed characters such as <tag>this & that</tag>).
- Inline—a, abbr, acronym, bdo, br, button, cite, code, del, dfn, em, img, input, ins, kbd, label, map, object, q, samp, select, span, strong, textarea, and var elements.
- Block—address, blockquote, div, dl, form, h1, h2, h3, h4, h5, h6, noscript, ol, p, pre, script, table, and ul elements.

For example, if the content is described as "Text, inline, or none," *<tag>this</tag>* (text) is valid, *<tag>this</tag>* (an inline element) is valid, *<tag></tag>* (none) is valid but *<tag> this </tag>* is not (because p is a block-level element). If the content is described as "One or more block," *<tag>this</tag>* is not valid (because it's just text inside), *<tag>this</tag>* is not valid

(because there's just an inline element inside), *<tag></tag>* is not valid (because there's nothing inside) but *<tag> this </tag>* is (because there's a block-level element inside).

<a>

Anchor. Primarily used as a hypertext link.

See Chapter 3, "Links."

Attributes

- [Common attributes]
- href—The target of the link. The value of the href attribute can be any URI, such as a web page, a directory, a file, or a page anchor.
- charset—The character set of the target of the link.
- type—The MIME type of the target of the link.
- hreflang—The language (in the form of a language code, such as "en" or "fr") of the target of the link. It should only be used when href is also used.
- rel—The relationship of the target of the link to the current page.
- rev—The relationship of the current page to the target of the link.
- accesskey—Associates a keyboard shortcut to the element.
- tabindex—Where the element appears in the tab order of the page.

Content

Text, inline (not including a), or none.

Example

```
<p><a href="http://www.htmldog.com">Link to a URI</a></p>
<p><a href="#content">Link to a page anchor</a></p>
```

Related Tags

base

<abbr></abbr>

Abbreviation—a shortened form of a word or phrase. *HTML* is an abbreviation, as is *CSS*, for example. Not recognized by IE.

See Chapter 2, "Text."

Attributes

- [Common attributes]—Note the title attribute is generally used to specify the whole word or phrase that the abbreviation is referring to.

Content

Text, inline, or none.

Example

```
<p>Jiminy Locust was trying to learn <abbr title="HyperText Markup
Language">HTML</abbr> but unfortunately he was a <abbr title="Dumb
insect who couldn't comprehend what a computer was, let alone use
one">DIWCCWACWLAUO</abbr>.</p>
```

Related Tags

acronym

<acronym></acronym>

Acronym—a pronounceable abbreviation that is made up of the initial letters or parts of words of that phrase. *NATO* is an example of an acronym, as is *UNICEF*.

See Chapter 2, "Text."

Attributes

- [Common attributes]—Note the title attribute is generally used to specify the whole word or phrase that the acronym is referring to.

Content

Text, inline, or none.

Example

```
<p>Jiminy was launched into space in a <acronym title="National
Aeronautics and Space Administration">NASA</acronym> rocket.</p>
```

Related Tags

abbr

<address></address>

Very specifically intended to mark up the contact details, such as a street address, for a page, or major part of a page (such as a contact form).

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<address>HMTL Dog House<br />HTML Street<br />Dogsville<br />HT16
3ML</address>
```

Related Tags

[none]

<area />

A region of a client-side image map. Used in conjunction with `map` to map links to certain regions of an image.

Attributes

- [Common attributes]
- alt (required)—The alternative text of the area, which should be a short description.
- shape—The shape of the area; the value can be rect (rectangular), circle (circular), poly (polygonal), or default.
- coords—The pixel coordinates of the area. For rectangular shapes, this is a comma-separated list of four values for left, top, right, and bottom (e.g., coords="0,0,50,50"). For circular shapes this is a comma-separated list of three values for left, top, and radius (e.g., coords="50,50,25"). For polygonal shapes, this is a comma-separated list containing an even number of values, specifying the left and top of each point of the shape (e.g., coords="0,0,25,25,50,25,50,100").
- href—The target of the area link.
- nohref—Used to specify that the area is not a link. It must be used in the format nohref="nohref".
- accesskey—Associates a keyboard shortcut to the area.
- tabindex—Where the area appears in the tab order of the page.

Content

Empty.

Example

```
<map id ="atlas">
    <area shape ="rect" coords ="0,0,115,90" href ="northamerica.
html" alt="North America" />
```

```
    <area shape ="poly" coords ="113,39,187,21,180,72,141,77,117,86"
href ="europe.hmtl" alt="Europe" />
    <area shape ="poly" coords ="119,80,162,82,175,102,183,102,175,1
48,122,146" href ="africa.html" alt="Africa" />
</map>
```

Related Tags

`map`

<base />

The base location from which files should be accessed. Relative links within a document (such as `<a href="someplace.html"`... or `<img src="someimage.jpg"`...) will become relative to the path specified by the `base` element.

The `base` element must go inside the `head` element.

See Chapter 1, "Getting Started."

Attributes

- href (required)—The location of the base URI.

Content

Empty.

Example

```
<base href="/images/tootlepops/" />
```

If the above example is applied then every file reference in the page will be in relation to "/images/tootlepops/". So, for example, `<img src="banana.jpg" alt="banana" />` will actually point to "/images/tootlepops/banana.jpg" and `<a href="morefruit/cucumber.html">Cucumber</a>` will actually point to "/images/tootlepops/morefruit/cucumber.html".

Related Tags

`a`, `img`

<bdo></bdo>

Bi-directional text. Used to define different directional content to the rest of the content on a page, such as languages that are read in a different direction from the default language (Hebrew in an English document, for example).

See Chapter 2, "Text."

Attributes

- [Core attributes]
- dir (required)—The direction of the text; can be set to ltr (left-to-right) or rtl (right-to-left).
- xml:lang—The language of the text.

Content

Text, inline, or none.

Example

```
<bdo dir="rtl">smug desserts</bdo>
```

Related Tags

[none]

<blockquote></blockquote>

A large, usually standalone, block-level quotation.

See Chapter 2, "Text."

Attributes

- [Common attributes]
- cite—The location (in the form of a URI) where the quote has come from.

Content

One or more blocks. The content of a `blockquote` element must be made up of other block-level elements, which in practice would usually be `p` elements.

Example

```
<blockquote title="From HTML Dog, Chapter 2"><p>blockquote is
designed to be for large, stand-alone quotations, whereas q (quote)
is used for smaller inline quotes.</p></blockquote>
```

Related Tags

`q`, `cite`

<body></body>

The main body of an HTML document where all of the content is placed. This is the stuff that people will see, hear, or otherwise experience when they visit the web page. Required, funnily enough, and should be used just once. It must start immediately after the closing `head` tag and end directly before the closing `html` tag.

See Chapter 1, “Getting Started.”

Attributes

- [Common attributes]

Content

Block or none.

Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
    <title>Uncle Jack's Sea Cow Farm</title>
</head>
<body>
```

```
<!-- A whole load of content -->
</body>
</html>
```

Related Tags

head, html

Line break.

See Chapter 2, "Text."

Attributes

- [Core attributes]

Content

Empty.

Example

```
<p>Greetings one and all.<br />Welcome to the world of line
breaks.</p>
```

Related Tags

p

<button></button>

Defines a form button that has content within it.

See Chapter 9, "Forms."

Attributes

- [Common attributes]
- accesskey—Associates a particular keyboard shortcut to the element.

- tabindex—Where the element appears in the tab order of the page.
- disabled—Disables the button. It must be used in the format disabled="disabled".
- name—Associates a name to the button so that it can be processed by a form-handling script.
- type—The button type. Values can be button (doesn't do anything), submit (default; submits the form when the button is selected), or reset (resets the form).
- value—An initial value that will appear as the button's label.

Content

Text, block (not including **form** or **fieldset**), inline (not including **input**, **select**, **textarea**, **label**, or **button**), or none.

Example

```
<button>Push my <strong>button</strong> baby</button>
```

Related Tags

input, **form**

<caption></caption>

A caption for a table. This should be placed directly after the opening **table** tag and will be displayed above the table by default.

See Chapter 8, "Tables."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<table>
    <caption>Animal groups</caption>
    <!-- etc. -->
</table>
```

Related Tags

`table`

<cite></cite>

In-line citation or reference to another source.

See Chapter 2, "Text."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<p>So I asked <cite>Bob</cite> about quotations and he said <q>I
know as much about quotations as I do about pigeon fancying</q>.
Luckily, I found 'HTML Dog' and it said…</p>
```

Related Tags

`q`, `blockquote`

<code></code>

Code, such as computer code.

See Chapter 2, "Text."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<code>norahjonesisbland=true;</code>
```

Related Tags

`samp`, `var`, `pre`

<col />

Table column. Allows attributes to be applied to a table column. Must be used within a `colgroup` element.

See Chapter 8, "Tables."

Attributes

- [Common attributes]
- span—The number of columns the element applies to.

Note: There are other valid attributes (`align`, `valign`, `char`, `charoff`) but they are presentational and so CSS should be used instead.

Content

Empty.

Example

```
<table>
    <colgroup>
        <col />
        <col class="alternative" />
```

```
        <col />
    </colgroup>
    <tr>
        <th>Cats</th>
        <th>Dogs</th>
        <th>Lemurs</th>
    </tr>
    <!-- etc. -->
</table>
```

Here, the styles of the class "alternative" will be applied to the second column, i.e., the second cell in every row.

Related Tags

`colgroup`, `tr`

<colgroup></colgroup>

Table column group. Allows attributes to be applied to a set of table columns.

See Chapter 8, "Tables."

Attributes

- [Common attributes]
- span—The number of columns the element applies to.

Note: There are other valid attributes (`align`, `valign`, `char`, `charoff`) but they are presentational and so CSS should be used instead.

Content

`col` elements or none

Example

```
<table>
    <colgroup span="2" class="alternative"></colgroup>
```

```
    <tr>
        <th>Cats</th>
        <th>Dogs</th>
        <th>Lemurs</th>
    </tr>
    <!-- etc. -->
</table>
```

Related Tags

`col`, `tr`

<dd></dd>

A definition description that is paired with one or more definition terms within a definition list.

See Chapter 6, "Lists."

Attributes

- [Common attributes]

Content

Text, block, inline, or none.

Example

```
<dl>
    <dt>Dog</dt>
    <dd>A carnivorous mammal of the family Canidae.</dd>
</dl>
```

Related Tags

`dl`, `dt`

<del></del>

An editorial deletion. Used in conjunction with `ins` when you want to track changes in a document.

See Chapter 2, "Text."

Attributes

- [Common attributes]
- cite—The location (as a URI) of an explanation of why the insertion was made.
- datetime—When the deletion was made (in the format of YYYYMMDD).

Content

Text, block, inline, or none.

Example

```
<p>Patrick was walking down the road when he saw a <del datetime="2
0040329">fluffy kitten</del><ins cite="http://www.htmldog.com">giant
rabid snarling mutant saber-toothed goat</ins>.</p>
```

Related Tags

`ins`

<dfn></dfn>

Definition term.

See Chapter 2, "Text."

Attributes

- [Common attributes]—Note the title attribute is often used to describe the definition.

Content

Text, inline, or none.

Example

```
<p><dfn title="Microsoft web browser">Internet Explorer</dfn> is the
most popular browser used underwater.</p>
```

Related Tags

`abbr`

Division. A block-level element that groups together a multiple HTML elements. Commonly used to apply CSS to a chunk of a page.

See Chapter 1, “Getting Started,” and Chapter 5, “Layout.”

Attributes

- [Common attributes]

Content

Text, block, inline, or none.

Example

```
<div id="content">
    <h1>How to make a falafel</h1>
    <p>Buy a falafel seed and plant it in your garden.</p>
</div>
```

Related Tags

`span`

<dl></dl>

Definition list, which contains terms and descriptions.

See Chapter 6, "Lists."

Attributes

- [Common attributes]

Content

One or more `dt` or `dd`.

Example

```
<dl>
    <dt>Cat</dt>
    <dd>A little furry thing that purrs.</dd>
    <dt>Dog</dt>
    <dd>A big shaggy thing that barks.</dd>
</dl>
```

Related Tags

`dt`, `dd`, `ul`

<dt></dt>

A definition term that is paired with one or more definition descriptions within a definition list.

See Chapter 6, "Lists."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<dl>
    <dt>Dog</dt>
    <dd>A carnivorous mammal of the family Canidae.</dd>
</dl>
```

Related Tags

`dl`, `dd`

Emphasis.

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<p>You lookin' at <em>me</em>?</p>
```

Related Tags

`strong`

<fieldset></fieldset>

A group of related form items.

See Chapter 9, "Forms."

Attributes

- [Common attributes]

Content

Text, `legend`, block, inline, or none.

Example

```
<form action="whatever.php">
    <fieldset>
        <!-- lots of form fields -->
    </fieldset>
    <fieldset>
        <!-- lots of form fields -->
    </fieldset>
</form>
```

Related Tags

`form`, `legend`

<form></form>

A form, allowing the sending of user-input data.

See Chapter 9, "Forms."

Attributes

- [Common attributes]
- action (required)—Tells the browser where to send the form data when it is submitted. This can be any URI, the destination of which will be a script where the form data is initially processed.
- method—Tells the browser how to send the form data. You have two options here: get or post.
- enctype—The MIME type used to encode the form data. The default value is application/x-www-form-urlencoded, but this should be multipart/form-data when the form contains a file input element.
- accept—Which file-types (selected from a file input element) should be accepted. This is a comma-separated list of MIME types.
- accept-charset—Which character sets should be accepted. This is a comma-separated list.

Content

One or more block (not including `form`) or `fieldset`.

Example

```
<form action="processor.php" method="post">
    <!-- a whole load of form fields -->
</form>
```

Related Tags

`input`, `fieldset`, `label`

<h1></h1>, <h2></h2>, <h3></h3>, <h4></h4>, <h5></h5>, <h6></h6>

Heading 1 (highest level heading) to Heading 6 (lowest level subheading). Headings should be used in order and `h1` used just once.

See Chapter 2, “Text.”

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<h1>Headings</h1>
<p>This is all about headings.</p>
<h2>The First Subheading</h2>
<p>The first subheading was called Bob. Bob was a figurine cleaner
in a past life.</p>
<h2>The Second Subheading</h2>
<p>The second subheading was called Labella. She used to be a
chimney sweep.</p>
<h3>Labella's Chimney Sweeping</h3>
<p>Labella can still be persuaded to sweep chimneys for five beans a
chimney.</p>
```

```
<h2>The Third Subheading</h2>
<p>The third subheading was called John. He wasn't particularly
interesting.</p>
```

Related Tags

`p`

<head></head>

The header of an HTML document where information about the document (rather than page content) is placed.

You must use this element and it should be used just once. It must start immediately after the opening `html` tag and end directly before the opening `body` tag.

See Chapter 1, "Getting Started."

Attributes

- [I18n attributes]
- profile—The location of information about the document. The value can be a URI or a number of URIs separated by spaces.

Content

Must include single `title`. Can include `base`, `link`, `meta`, `script`, and `style`.

Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
    <title>Uncle Jack's Sea Cow Farm</title>
</head>
<body>
<!-- A whole load of content -->
</body>
</html>
```

Related Tags

body, html, title

<html></html>

The root element that specifies that the content of the document is HTML. It contains all of the remainder of the page information after the document type declaration.

See Chapter 1, "Getting Started."

Attributes

- [I18n attributes]
- xmlns (required)—The XML namespace. The value must be http://www.w3.org/1999/xhtml.

Content

One head and one body.

Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
    <title>Uncle Jack's Sea Cow Farm</title>
</head>
<body>
<!-- A whole load of content -->
</body>
</html>
```

Related Tags

head, body

An image.

See Chapter 4, "Images."

Attributes

- [Common attributes]
- src (required)—The location of the image file.
- alt (required)—The alternative text of the image. This provides placeholder text while the image is downloading. It also serves an important accessibility task: It provides an "alternative" to the image for those who cannot see the image itself.
- longdesc—The location (in the form of a URI) of a description of the image. An accessibility consideration, used for detailed images containing important content (such as a map or a chart).
- height—The height of the image (in pixels).
- width—The width of the image (in pixels).

Note: **border** can also be used, although using CSS is preferable.

Content

Empty.

Example

```
<img src="images/sifaka.jpg" alt="Leaping sifaka" />
```

Related Tags

[none]

<input />

A form field that can be represented as a text box, password text box, checkbox, radio button, submit button, reset button, hidden field, image, file selection box, or general button.

See Chapter 9, "Forms."

Attributes

- [Common attributes]
- name—Provides an identifier for the element's data.
- type—The type of input. Values can be text (default), password, checkbox, radio, submit, reset, hidden, image, file, or button.
- value—The initial value. It is required when type is set to checkbox or radio. It should not be used when type is set to file.
- checked—For type="checkbox" or type="radio", sets the initial state to selected. Used in the format checked="checked".
- maxlength—Sets a limit on the number of characters allowed in a text box.
- src—For type="image", specifies the location of the image file.
- alt—For type="image", specifies the alternative text of the image.
- accept—For type="file", specifies which file types should be accepted. This is a comma-separated list of MIME types.
- disabled—Disables an element. Used in the format disabled="disabled".
- readonly—Specifies that the value of the element cannot be changed. Used in the format readonly="readonly".
- accesskey—Associates a keyboard shortcut to the element.
- tabindex—Specifies where the element appears in the tab order of the page.

Content

Empty.

Example

```
<form action="somescript.php" />
    <p>Do you like pie?</p>
    <div>yes <input type="radio" name="pie" value="yes"
checked="checked" /></div>
    <div>no <input type="radio" name="pie" value="no" /></div>
    <div>Your name: <input type="text" name="yourname" /></div>
    <div><input type="image" name="submitimage" src="someimage.gif"
/></div>
</form>
```

Related Tags

`form`, `textarea`, `select`, `label`

<ins></ins>

An editorial insertion. Used in conjunction with `del` when you want to track changes in a document.

See Chapter 2, "Text."

Attributes

- [Common attributes]
- cite—The location (as a URI) of an explanation of why the insertion was made.
- datetime—When the insertion was made (in the format of YYYYMMDD).

Content

Text, block, inline, or none.

Example

```
<p>Patrick was walking down the road when he saw a <del datetime="2
0040329">fluffy kitten</del><ins cite="http://www.htmldog.com">giant
rabid snarling mutant saber-toothed goat</ins>.</p>
```

Related Tags

`del`

<kbd></kbd>

Keyboard. Used to specifically suggest text that should be entered by the user.

See Chapter 2, "Text."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<p>Now type <kbd>banana</kbd>.</p>
```

Related Tags

`code`

<label></label>

Label for a form element (`input`, `textarea`, or `select`).

See Chapter 9, "Forms."

Attributes

- [Common attributes]
- for—Associates the label to a form element when the value of for matches the value of an element's id attribute.
- accesskey—Associates a keyboard shortcut to the element.

Content

Text, inline (not including `label`), or none.

Example

```
<label for="email">Email address</label><input type="text"
name="email" id="email" />
```

Related Tags

`input`, `textarea`, `select`

<legend></legend>

Defines a caption for a `fieldset`. The element must appear directly after the opening `fieldset` tag.

Attributes

- [Common attributes]
- accesskey—Associates a keyboard shortcut to the element.

Content

One or more text or inline.

Example

```
<fieldset>
    <legend>Book Details</legend>
    <!-- lots of form fields -->
</fieldset>
```

Related Tags

`fieldset`

List item. An item in any `ul` or `ol` element.

See Chapter 6, “Lists.”

Attributes

- [Common attributes]

Content

Text, block, inline, or none.

Example

```
<ul>
    <li>This</li>
    <li>That</li>
    <li>The other</li>
</ul>
```

Related Tags

`ul`, `ol`

<link />

Defines a link to an external resource such as a CSS file, a shortcut icon, or customized navigation.

See Chapter 1, "Getting Started."

Attributes

- [Common attributes]
- href—The target of the link.
- charset—The character set of the target of the link.
- hreflang—The language of the target of the link.
- type—The MIME type of the target of the link.
- rel—The relationship of the target of the link to the current page. Some universally understood values are shortcut icon and stylesheet.
- rev—The relationship of the current page to the target of the link.

- media—Which media the link is associated to. A value such as screen, print, projection, braille, speech, or all can be used or a combination in a comma-separated list.

Content
Empty.

Example
```
<link rel="stylesheet" type="text/css" title="Some title" href="/
somefile.css" />
<link rel="alternate stylesheet" type="text/css" title="Some
alternative title" href="/someotherfile.css" />
<link rel="shortcut icon" href="/favicon.ico" /><link rel="next"
title="Next page" href="nextpage.html" />
```

Related Tags
head

<map></map>

A client-side image map. Used in conjunction with area to map links to certain regions of an image.

Attributes
- [I18n attributes]
- [Events attributes]
- id (required)—Uniquely identifies the element.
- class—Used to reference the element with CSS.
- title—A title for the element.

Content
One or more blocks or areas.

Example

```
<map id="atlas">
    <area shape ="rect" coords ="0,0,115,90" href ="northamerica.
html" alt="North America" />
    <area shape ="poly" coords ="113,39,187,21,180,72,141,77,117,86"
href="europe.hmtl" alt="Europe" />
    <area shape ="poly" coords ="119,80,162,82,175,102,183,102,175,1
48,122,146" href ="africa.html" alt="Africa" />
</map>
```

Related Tags

area

<meta />

Meta information. Used to provide information about the HTML page.

See Chapter 1, “Getting Started.”

Attributes

- [I18n attributes]
- content (required)—The meta information itself.
- name—The name given to the meta information. Frequently used values of the name attribute are “keywords” and “description,” but they can be absolutely anything.
- http-equiv—Used to define an “equivalent” HTTP header for the document when name is not used.
- scheme—Specifies how the value of content should be interpreted.

Content

Empty.

Example

```
<meta name="keywords" content="fruit, banana, orange, apple,
kumquat, cucumber" />
<meta name="description" content="News, reviews and opinion on all
things fruity." />
<meta name="author" content="The Fruit Farmers Association of
Bujumburra" />
<meta name="date" scheme="Day-Month-Year" content="12-01-99" />
```

Related Tags

head

<noscript></noscript>

Content to be used when scripts cannot be executed, through browser inadequacies or user choice.

See Chapter 7, "Scripts & Objects."

Attributes

- [Common attributes]

Content

Block.

Example

```
<noscript>
<p>What? No JavaScript? Well what am I supposed to do now? Can't you
get a new browser or something?</p>
</noscript>
```

Related Tags

script

<object></object>

An embedded multimedia object such as a movie or a sound file.

See Chapter 7, "Scripts & Objects."

Attributes

- [Common attributes]
- data—The location of the data for the object in the form of a URL.
- type—The content type of the data specified by the data attribute. This basically lets the browser know what kind of file to expect.
- declare—Specifies that the object is a declaration only. Must be used in the format declare="declare".
- classid—The location of the object in the form of a URL or Windows Registry location.
- codebase—The base location from which relative URLs specified in the classid, data, and archive attributes should be taken.
- codetype—The content type of the object.
- archive—Resources relevant to the object. The value should be a URL or a number of URLs separated by spaces.
- standby—Text that will be displayed while the object is loading.
- height—The height of the object (in pixels), just like in an img element.
- width—The width of the object (in pixels), again, just like in an img element.
- name—A name by which the object can be referenced.
- tabindex—Where the element appears in the tab order of the page.

Content

Text, block, inline, **param**, or none.

Example

```
<object type="blueberry/kumquat" data="whatever.kmq">
    <param name="tangy" value="true" />
    <param name="segments" value="9" />
    <p>You don't have the Kumquat plugin, so you won't get any
juice.</p>
</object>
```

Related Tags

`param`

Ordered list, suggesting that each item is in some way lower or higher than the item before or after it.

See Chapter 6, “Lists.”

Attributes

- [Common attributes]

Content

One or more `li`.

Example

```
<ol>
    <li>The first thing</li>
    <li>The second thing</li>
    <li>The third thing</li>
</ol>
```

Related Tags

`li`, `ul`, `dl`

<optgroup></optgroup>

Option group. Defines a group of `option` elements in a `select` form field.

See Chapter 9, "Forms."

Attributes

- [Common attributes]
- label (required)—Assigns a label to the option group.
- disabled—Disables an element. It must be used in the format disabled="disabled".

Content

One or more `option`.

Example

```
<select name="book">
    <optgroup label="Camus">
        <option>The Outsider</option>
        <option>The Rebel</option>
        <option>The Plague</option>
    </optgroup>
    <optgroup label="Orwell">
        <option>Animal Farm</option>
        <option>Nineteen Eighty-Four</option>
        <option>Down and Out in Paris and London</option>
    </optgroup>
</select>
```

Related Tags

`option`, `select`

<option></option>

Defines an option of a `select` form field.

See Chapter 9, "Forms."

Attributes

- [Common attributes]
- value—A value for the option. If value is not used, the value of the option element is set to its contents by default.
- selected—Used to specify that the option is initially selected. It must be used in the format selected="selected".

Content

Text

Example

```
<select name="dogs">
    <option>Domestic Dog</option>
    <option>Arctic Fox</option>
    <option>Maned Wolf</option>
    <option>Grey Wolf</option>
    <option>Red Fox</option>
    <option>Fennec</option>
</select>
```

Related Tags

select, optgroup

Paragraph.

See Chapter 2, "Text."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<p>Greetings, one and all. Welcome to the world of paragraphs.</p>
<p>This will be the second paragraph then…</p>
```

Related Tags

`h1` to `h6`, `em`, `strong`

<param />

Parameter of an `object`. It is often the case that you will want, or need, to pass certain parameters to the object.

See Chapter 7, "Scripts & Objects."

Attributes

- name (required)—Used so that the element can be referenced and processed by the object.
- value—The value of the parameter. The values of the name and value attributes are completely dependent on the object. All that param elements do is tell the object "I want to set this [name] to this [value]."
- id—Uniquely identifies the element.
- type—The content type.
- valuetype—The content type of the value attribute. Values can be data, ref, or object.

Content

Empty.

Example

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/
swflash.cab#version=6,0,0,0" width="200" height="300" id="penguin">
    <param name="movie" value="flash/penguin.swf" />
```

```
    <param name="quality" value="high" />
    <img src="images/penguin.jpg" width="200" height="300"
alt="Penguin" />
</object>
```

Related Tags

`object`

<pre></pre>

Preformatted text. Text where the white space (which is normally discarded by other elements) is as much a part of the content as the rest of the text.

See Chapter 2, "Text."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<pre>
&lt;div id="intro"& gt;
    &lt;h1&gt;Some heading&lt;/h1&gt;
     &lt;p&gt;Some paragraph paragraph thing thing thingy.&lt;/p&gt;
&lt;/div&gt;
</pre>
```

Related Tags

`code`

<q></q>

In-line quote. Used for small quotations.

See Chapter 2, "Text."

Attributes

- [Common attributes]
- cite—The location (in the form of a URI) where the quote has come from.

Content

Text, inline, or none.

Example

```
<p>So I asked Bob about quotations and he said <q>I know as much about
quotations as I do about pigeon fancying</q>. Luckily, I found 'HTML
Dog: The Best-Practice Guide to XHTML and CSS' and it said...</p>
```

Related Tags

`blockquote`, `cite`

<samp></samp>

Sample. Defines sample output, from a computer program, for example.

See Chapter 2, "Text."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<p>The result will either be <samp>Kid</samp> or <samp>Koala</
samp>.</p>
```

Related Tags

`code`

<script></script>

Defines a block of script. The tool of choice for inserting or loading a chunk of JavaScript into an HTML page.

See Chapter 7, "Scripts & Objects."

Attributes

- type (required)—The MIME type of the scripting language is used, such as text/javascript.
- src—An external source (URI) of a script file.
- charset—The character set of the element.
- defer—Used to specify that the script does not generate any document content so that the browser doesn't have to worry about it while the page loads. Must be used in the format defer="defer".

Content

Text (script) or none.

Example

The script itself can be placed between the opening and closing script tags, like so:

```
<script type="text/javascript">
function satsuma() {
    alert("SAAAATSUUUUMAAAA!!!");
}
</script>
```

Alternatively, a script can be kept in a separate file and applied like so:

```
<script type="text/javascript" src="kumquat.js"></script>
```

Related Tags

noscript

<select></select>

A drop-down list form element; `option` elements within the `select` element define each list item.

See Chapter 9, "Forms."

Attributes

- [Common attributes]
- name—Used so the value of the element can be processed.
- size—How many items of the list are displayed at any time. The default is `1`.
- multiple—Used to specify that more than one item from the list can be selected. This must be used in the format multiple="multiple".
- disabled—Disables an element. It must be used in the format disabled="disabled".
- tabindex— Where the element appears in the tab order of the page.

Content

One or more `optgroup` or `option`.

Example

```
<select name="book">
    <option>The Trial</option>
    <option>The Outsider</option>
    <option>Things Fall Apart</option>
    <option>Animal Farm</option>
</select>
```

Related Tags

`option`, `form`, `input`

An inline element that groups together a chunk of inline HTML, such as single words or short phrases. Commonly used to apply CSS to a small group of inline HTML elements.

See Chapter 1, "Getting Started."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<h1>How to make a <span>falafel</span></h1>
```

Related Tags

`div`

Strong emphasis.

See Chapter 2, "Text."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<p>You lookin' at <em>me</em>? You lookin' at <strong>me</strong>?
</p>
```

Related Tags

em

<style></style>

Used to define CSS at a page level. This sits inside the head element and its contents are simply a big ol' list of CSS rules.

See Chapter 1, "Getting Started."

Attributes

- [I18n attributes]
- type (required)—The content type, which is generally text/css.
- media—Which media the styles are associated to. The value can be aural, braille, embossed, handheld, print, projection, screen, tty (teletype), or tv (television). You could also have media="all", but that's the same as not having any media attribute at all.
- title—Assigns a title to the styles within the element. This can then be referenced by browsers or scripting languages to either disable the styles or switch between alternate style sheets.

Content

Text (CSS).

Example

```
<head>
    <title>Bujumburra</title>
    <style type="text/css">
    body {
        font-family: arial, Helvetica, sans-serif;
        color: black
    }
    /* etc. etc. */
    </style>
</head>
```

Related Tags

`head`, `link`

<table></table>

A table, used for tabular data.

See Chapter 8, "Tables."

Attributes

- [Common attributes]
- summary—A summary of the data represented in the table.

Note: There are other valid attributes (`border`, `cellpadding`, `cellspacing`, `frame`, `rules`, `width`) but they are presentational and so CSS should be used instead.

Content

Must have one or more `tr` or [single `thead`, single `tfoot`, or one or more `tbody`]. Can also have `col` or `colgroup` elements and a single `caption`.

Example

```
<table summary="The results of an inane quiz">
    <tr>
        <th>Question</th>
        <th>Answer</th>
        <th>Correct?</th>
    </tr>
    <tr>
        <td>What is the capital of Burundi?</td>
        <td>Bujumburra</td>
        <td>Yes</td>
    </tr>
    <tr>
        <td>Which came first, the chicken or the egg?</td>
        <td>The chicken</td>
        <td>No</td>
```

```
    </tr>
    <!-- etc. -->
</table>
```

Related Tags

`tr`, `td`

<tbody></tbody>

Table body row group. Can be used more than once, and must be used if `thead` or `tfoot` are used. It must be used within a `table` element and must follow both `thead` and `tfoot` elements when used.

See Chapter 8, "Tables."

Attributes

- [Common attributes]

Note: There are other valid attributes (`align`, `valign`, `char`, `charoff`) but they are presentational and so CSS should be used instead.

Content

One or more `tr`.

Example

```
<table>
<thead>
    <tr>
        <th>Header 1</th>
        <th>Header 2</th>
        <th>Header 3</th>
    </tr>
</thead>
<tfoot>
    <tr>
        <td>Footer 1</td>
        <td>Footer 2</td>
```

```
            <td>Footer 3</td>
        </tr>
    </tfoot>
    <tbody>
        <tr>
            <td>Cell data 1</td>
            <td>Cell data 2</td>
            <td>Cell data 3</td>
        </tr>
        <tr>
            <td>Cell data 4</td>
            <td>Cell data 5</td>
            <td>Cell data 6</td>
        </tr>
        <tr>
            <td>Cell data 7</td>
            <td>Cell data 8</td>
            <td>Cell data 9</td>
        </tr>
    </tbody>
    </table>
```

Related Tags

`tfoot`, `thead`, `table`

<td></td>

Table data cell. Must appear within a `tr` element.

See Chapter 8, "Tables."

Attributes

- [Common attributes]
- colspan—Specifies across how many columns the cell should spread. The default value is `1`.
- rowspan—Specifies across how many rows the cell should spread. The default value is `1`.

- abbr—An abbreviated version of the content of the cell.
- headers—Explicitly specifies which header cells are associated to the cell. The value is a single or comma-separated list of table cell id values.
- scope—Explicitly specifies that the cell contains header information for the rest of the row (value row), column (value col), row group (value rowgroup), or column group (value colgroup) that contains it.
- axis—A category that forms a conceptual axis in n-dimensional space for hierarchical structuring. The value can be a single name or a comma-separated list of names.

Note: There are other valid attributes (**align**, **valign**, **char**, **charoff**) but they are presentational and so CSS should be used instead.

Content

Text, block, inline, or none.

Example

```
<table>
    <tr>
        <td>Cats</td>
        <td>Dogs</td>
        <td>Lemurs</td>
    </tr>
    <tr>
        <td>Tiger</td>
        <td>Grey wolf</td>
        <td>Indri</td>
    </tr>
    <tr>
        <td>Cheetah</td>
        <td>Cape hunting dog</td>
        <td>Sifaka</td>
    </tr>
</table>
```

This example shows a table with three rows with three cells in each row, making it a 3x3 table.

Related Tags

tr, th, table

<textarea></textarea>

Creates a multiline text area form field. The initial value of the text area can be placed in between the opening and closing tags.

See Chapter 9, "Forms."

Attributes

- [Common attributes]
- rows (required)—The number of viewable rows.
- cols (required)—The number of viewable columns.
- name—Used so that the value of the element can be processed.
- disabled—Disables an element. It must be used in the format disabled="disabled".
- readonly—Used to specify that the value of the element cannot be changed. It must be used in the format readonly="readonly".
- accesskey—Associates a keyboard shortcut to the element.
- tabindex—Where the element appears in the tab order of the page.

Content

Text.

Example

```
<form action="somescript.php" />
    <p>Your address</p>
```

```
    <div><textarea name="address" cols="30" rows="4"></textarea></
div>
    <div><input type="submit" /></div>
</form>
```

Related Tags

`input`, `form`

<tfoot></tfoot>

Table footer row group. Along with `thead` and `tbody`, `tfoot` can be used to group a series of rows. `tfoot` can be used just once within a `table` element and must appear before a `tbody` element.

See Chapter 8, "Tables."

Attributes

- [Common attributes]

Note: There are other valid attributes (`align`, `valign`, `char`, `charoff`) but they are presentational and so CSS should be used instead.

Content

One or more `tr`.

Example

```
<table>
<thead>
    <tr>
        <th>Header 1</th>
        <th>Header 2</th>
        <th>Header 3</th>
    </tr>
</thead>
<tfoot>
    <tr>
```

```
            <td>Footer 1</td>
            <td>Footer 2</td>
            <td>Footer 3</td>
        </tr>
    </tfoot>
    <tbody>
        <tr>
            <td>Cell data 1</td>
            <td>Cell data 2</td>
            <td>Cell data 3</td>
        </tr>
        <tr>
            <td>Cell data 4</td>
            <td>Cell data 5</td>
            <td>Cell data 6</td>
        </tr>
        <tr>
            <td>Cell data 7</td>
            <td>Cell data 8</td>
            <td>Cell data 9</td>
        </tr>
    </tbody>
    </table>
```

Related Tags

`thead`, `tbody`, `table`

<th></th>

Table header cell. Must appear within a `tr` element.

See Chapter 8, "Tables."

Attributes

- [Common attributes]
- colspan—Specifies across how many columns the cell should spread. The default value is `1`.

- rowspan—Specifies across how many rows the cell should spread. The default value is 1.
- abbr—An abbreviated version of the content of the cell.
- headers—Explicitly specifies which header cells are associated to the cell. The value is a single or comma-separated list of table cell id values.
- scope—Explicitly specifies that the cell contains header information for the rest of the row (value row), column (value col), row group (value rowgroup), or column group (value colgroup) that contains it.
- axis—A category that forms a conceptual axis in n-dimensional space for hierarchical structuring. The value can be a single name or a comma-separated list of names.

Note: There are other valid attributes (**align**, **valign**, **char**, **charoff**) but they are presentational and so CSS should be used instead.

Content

Text, block, inline, or none.

Example

```
<tr>
    <th>Cats</th>
    <th>Dogs</th>
    <th>Lemurs</th>
</tr>
```

Table header cells can also be used as headers for rows, for example if you had your table structured like this:

```
<table>
    <tr>
        <th>Cats</th>
        <td>Tiger</td>
        <td>Cheetah</td>
    </tr>
    <tr>
```

```
        <th>Dogs</th>
        <td>Grey wolf</td>
        <td>Cape hunting dog</td>
    </tr>
    <tr>
        <th>Lemurs</th>
        <td>Indri</td>
        <td>Sifaka</td>
    </tr>
</table>
```

Related Tags

`td`, `tr`, `table`

<thead></thead>

Table header row group. Along with `tfoot` and `tbody`, `thead` can be used to group a series of rows. `thead` can be used just once within a `table` element and should appear before a `tfoot` and `tbody` element.

See Chapter 8, "Tables."

Attributes

- [Common attributes]

Note: There are other valid attributes (`align`, `valign`, `char`, `charoff`) but they are presentational and so CSS should be used instead.

Content

One or more `tr`.

Example

```
<table>
<thead>
<tr>
    <td>Header 1</td>
```

```
        <td>Header 2</td>
        <td>Header 3</td>
    </tr>
    </thead>
    <tfoot>
    <tr>
        <td>Footer 1</td>
        <td>Footer 2</td>
        <td>Footer 3</td>
    </tr>
    </tfoot>
    <tbody>
    <tr>
        <td>Cell 1</td>
        <td>Cell 2</td>
        <td>Cell 3</td>
    </tr>
    <!-- etc. -->
    </tbody>
    </table>
```

Related Tags

`tfoot`, `tbody`, `table`

<title></title>

This simply gives a title to the HTML document. It will appear as the title of the browser window, and is also used for bookmarks. It is required and must be placed within the `head` element.

See Chapter 1, “Getting Started.”

Attributes

- [I18n attributes]

Content

Text.

Example

```
<head>
    <title>Uncle Jack's Sea Cow Farm</title>
</head>
```

Related Tags

`head`

<tr></tr>

Table row. Must appear within a `table` element.

See Chapter 8, "Tables."

Attributes

- [Common attributes]

Note: There are other valid attributes (`align`, `valign`, `char`, `charoff`) but they are presentational and so CSS should be used instead.

Content

One or more `td` or `th`.

Example

```
<table>
    <tr>
        <th>Question</th>
        <th>Answer</th>
        <th>Correct?</th>
    </tr>
    <tr>
        <td>What is the capital of Burundi?</td>
```

```
        <td>Bujumburra</td>
        <td>Yes</td>
    </tr>
    <tr>
        <td>Which came first, the chicken or the egg?</td>
        <td>The chicken</td>
        <td>No</td>
    </tr>
    <!-- etc. -->
</table>
```

Related Tags

`td`, `table`

Unordered list. As its name implies, an unordered list is for non-ordinal items, in which any item could feel just as at home at one point in the list as any other.

See Chapter 6, "Lists."

Attributes

- [Common attributes]

Content

One or more `li`.

Example

```
<ul>
    <li>This</li>
    <li>That</li>
    <li>The Other</li>
</ul>
```

Related Tags

`li`, `ol`, `dl`

<var></var>

A variable in computer code.

See Chapter 2, "Text."

Attributes

- [Common attributes]

Content

Text, inline, or none.

Example

```
<code><var>norahjonesisbland</var>=true;</code>
```

Related Tags

code

Bad Tags

In ancient texts you may read of the twisted mythology of tags that that have no place in the real world. Bad Tags usually come down to tags that are presentational, which is the realm of CSS, or simply not valid, leading to unreliable code that can't be guaranteed to work on different or future browsers. See this book's Introduction for more on why such tags should be avoided.

You can also read a more detailed explanation of why the following are Bad Tags at *www.htmldog.com/guides/htmlintermediate/badtags*.

- applet—Embed a Java applet. Not valid. Use object tag.
- b—Bold. A valid tag, but purely presentational. Use CSS font-weight for bold or HTML em or strong tags for emphasis.
- big—Big text. A valid tag, but purely presentational. Use CSS font-size.

- blink—Blinking text. Not valid. Use JavaScript or CSS text-decoration: blink if you really insist on inflicting this.
- center—Center. Not valid. Use CSS margin: 0 auto or text-align: center.
- embed—Embed a multimedia object. Use object tag.
- font—Font name and size. Not valid. Use CSS font, font-family, and font-size.
- frame, frameset, iframe—Frames. Not valid. Framesets can be established with a different XHTML Doctype (see Chapter 1, "Getting Started"). Future standards (Xframes) dictate that frames should be completely separate from HTML, reducing usability and accessibility problems. CSS position: fixed can replicate some features of frames. JavaScript could also be used.
- hr—Horizontal rule. A valid tag, but presentational. Perhaps the most controversial of these Bad Tags, many argue that this has a genuine role as a divider of content. As it belongs to the XHTML Presentation Module and as its name implies, however, to truly separate structure and presentation, hr should be avoided. CSS borders can replicate horizontal rules, as can background images.
- i—Italic. A valid tag, but purely presentational. Use CSS font-style for italics or HTML em or strong tags for emphasis.
- layer—Layer. Not valid. Use HTML div and CSS position.
- marquee—Scrolling text. Not valid. Use JavaScript, Flash, or other plugin.
- small—Small text. A valid tag, but purely presentational. Use CSS font-size.
- sub—Subscript. A valid tag, but purely presentational. Use CSS vertical-align or position.
- sup—Superscript. A valid tag, but purely presentational. Use CSS vertical-align or position.
- tt—Teletype. A valid tag, but purely presentational. Use CSS font courier or similar for appearance or HTML code tag for computer code.
- u—Underline. Not valid. Use CSS text-decoration.

Rotten Attributes

Rotten attributes are the evil little disciples of the Bad Tags. Like the Bad Tags, their crime is usually one of presentation or downright invalidity.

- align—Aligns content. Not valid. As with the center tag, CSS text-align should be used.
- background—Background image. Not valid. Use CSS background-image.
- link, alink, and vlink—Non-visited, active, and visited link colors. Not valid. Use CSS :link, :active, and :visited pseudo-classes.
- marginwidth, marginheight, topmargin, and leftmargin—Page margins (used in the opening body tag). Not valid. Use CSS margin or padding.
- name—Used to assign an identifying name to an element. Invalid for all elements apart from button, input, select, textarea, meta, param, and object. Use the id attribute.
- target—Specifies where a link should open (such as in a new window). Not valid. JavaScript is a possible alternative, but the use of this should be questioned due to the adverse effect it has on usability and accessibility.
- text and bgcolor—Text color and background color. Not valid. Use CSS color and background-color.

CSS Reference

THIS CSS REFERENCE covers the pseudo-classes, pseudo-elements, at-rules, and properties of CSS 2 revision 1 (with the exception of aural CSS).

Specific chapters are highlighted for cross-referencing when there is relevant extended information (which there will be for all but the less-practical aspects, such as those that are not widely supported). Browser support issues are also noted where relevant. "Not supported by IE" (Internet Explorer) comes up a fair bit, and relates to Internet Explorer versions 6 and earlier (IE 7 has fixed many of its predecessors' shortcomings).

More on the syntax and application of CSS can be found in Chapter 1, "Getting Started."

Pseudo-classes

:active

Applies declarations to a box that is being activated by the user (such as while the mouse button is pressed). IE 6 and below will only apply `:active` to `a` elements.

See Chapter 3, "Links."

Example

```
a:active { color: red; }
```

See Also

`:link`, `:visited`, `:hover`, `:focus`

:first

Applies declarations to the first page in paged media.

Example

```
@page:first { margin-top: 10cm; }
```

See Also

`:left`, `:right`

:first-child

Applies declarations to the box of the first instance of an element inside another element. Not supported by IE 6 or below.

Example

```
p em:first-child { font-weight: bold; }
```

See Also

[none]

:focus

Applies declarations to a box that receives focus. Not supported by IE.

See Chapter 3, "Links," and Chapter 9, "Forms."

Example

```
input:focus { background-color: yellow; }
```

See Also
`:link`, `:visted`, `:hover`, `:active`

:hover

Applies declarations when a box that is hovered over by the cursor. IE 6 and below will only apply `:hover` to `a` elements.

See Chapter 3, "Links."

Example
```
a:hover { text-decoration: none; }
```

See Also
`:link`, `:visited`, `:active`, `:focus`

:lang

Applies declarations to the boxes of elements of a specific language, which is specified in brackets following the selector. Not supported by any major browser.

Example
```
html:lang(fr) { color: green; }
```

See Also
[none]

:left

Applies declarations to left pages in paged media.

Example
```
@page:left { margin-left: 5cm; }
```

See Also

`:right`, `:first`

:link

Applies declarations to the box of a link, the destination of which *has not* been visited.

See Chapter 3, "Links."

Example

```
a:link { color: #009; }
```

See Also

`:visited`, `:hover`, `:active`, `:focus`

:right

Applies declarations to right pages in print media.

Example

```
@page:right { margin-right: 5cm; }
```

See Also

`:left`, `:first`

:visited

Applies declarations to the box of a link, the destination of which *has* been visited.

See Chapter 3, "Links."

Example

```
a:visited { color: #999 }
```

See Also

`:link`, `:hover`, `:active`, `:focus`

Pseudo-elements

:after

Inserts generated content after the displayed content of a box. Not supported by IE.

Example

```
p:after { content: url(pieface.jpg); }
```

See Also

`:before`

:before

Inserts generated content before the displayed content of a box. Not supported by IE.

Example

```
h2:before { content: "Chapter: "; }
```

See Also

`:after`

:first-letter

Applies declarations to the first character of text in a box. Not supported by IE/Win 5.0.

Example

```
p:first-letter { font-size: 2em; }
```

See Also
`:first-line`

:first-line

Applies the declarations to the first visible line of text in a box. Not supported by IE/Win 5.0.

Example
```
p:first-line { font-weight: bold; }
```

See Also
`:first-letter`

At-rules

@import

Imports rules from another style sheet into the current one.

The value can be a string or a string wrapped by `url()` and can be followed by a comma-separated list of the media types that the import should apply to. If no media types are stated, the rule will apply to all.

See Chapter 1, "Getting Started."

Example
```
@import url("poodle.css") print;
```

@media

Applies rules to a particular medium.

See Chapter 10, "Multiple Media."

Example

```
@media print {
     body { font: 10pt "times new roman", times, serif; }
     #navigation { display: none; }
}
```

@page

Applies declarations to paged media.

Example

```
@page { margin: 3cm; }
```

Properties

The possible values for a property will be take one of these formats (see Chapter 1, "Getting Started"):

- valuename—A straightforward keyword, such as `block` in `display: block`.
- valuename (default)—When a value is marked as "default" it means that this is the value that all XHTML elements will initially have.
- [length]—Such as `10em`, `300px`, `12pt`, `3cm`, etc.
- [percentage]—Such as `40%`.
- [color]—A hex value (such as `#f00`, or `#ff0000`), an RGB value (such as `rgb(255, 0, 0)`) or a color name (`aqua`, `black`, `blue`, `fuchsia`, `gray`, `green`, `lime`, `maroon`, `navy`, `olive`, `orange`, `purple`, `red`, `silver`, `teal`, `white`, and `yellow`. You can also use the value `transparent`.
- [URI]—File location such as `url(thingy.jpg)` or `url(http://www.whatever.com/whatever/whatever.gif)`.
- [number]—Such as `3` or `235` (no unit).

The ever-prolific but seldom used `inherit` value explicitly sets the same computed value as that of its parent element. Many properties inherit values by default (the ones you would normally want to be inherited), meaning the use of inherit is rarely necessary.

background

A "shorthand" property that combines `background-color`, `background-image`, `background-repeat`, `background-attachment`, and `background-position` in one handy property.

See Chapter 4, "Images."

Possible values

[A combination of some or all of the values for `background-color`, `background-image`, `background-repeat`, `background-attachment`, and `background-position`.]

Example

```
body {
    background: #0084c7 url(images/sifakabg.gif) top left fixed no-repeat;
}
```

Related Properties

`background-color`, `background-image`

background-attachment

Determines whether the background image should scroll with the content of a box.

See Chapter 4, "Images."

Possible values

- inherit

- scroll (default)—The background image will scroll when the rest of the content is scrolled.
- fixed—The background image will remain stationary when the rest of the content is scrolled.

Example

```
body {
    background-image: url(images/sifakabg.gif);
    background-attachment: fixed;
}
```

This example will plaster the "sifakabg.gif" image across the page and rather than the pattern scrolling, as it would do on a long page with lots of content in it, it will stick right where it is, with the rest of the page scrolling over the top.

Related Properties

`background`, `background-image`

background-color

Background color.

See Chapter 2, "Text," and Chapter 4, "Images."

Possible values

- inherit
- transparent (default)
- [color]

Example

```
body {
    font-family: "Times New Roman", Times, serif;
    color: white;
```

```
    background-color: black;
}
blockquote {
    background-color: #efe;
}
```

Related Properties

color, background

background-image

The background image of a box. The background-image property can be used to specify an image to be used as a background for just about any element box—from the page body to a paragraph to a link. Use it on its own, and the image will magically tile itself across the background of the element starting from the top left corner and repeating horizontally and vertically, filling the box.

See Chapter 4, "Images."

Possible values

- inherit
- none (default)
- [URI]

Example

```
body {
    background-image: url(images/sifakabg.gif);
}
```

Related Properties

color, background

background-position

The position of a background image within its box. Background images will start at the top left corner of a box by default, but you can change this with the `background-position` property, which is particularly useful when `background-repeat` is set to `no-repeat`, for example.

Possible values

- inherit
- top
- right
- bottom
- left
- [percentage]
- [length]
- [combination]—Such as `background-position: top left;`

Example

```
body    {
    background-image: url(images/sifakabg.gif);
    background-repeat: no-repeat;
    background-position: center;
}
```

Related Properties

`background-image`, `background`

background-repeat

How a background image will repeat itself. You don't have to have the background image tiled (repeated over and over, horizontally and vertically as space allows)—you

can decide whether you want it to repeat just horizontally, just vertically, or not at all by using the `background-repeat` property. Those areas of the element that are not taken up by the background image will be transparent, unless coupled with a background color, which would paint the rest of the area that color.

See Chapter 4, "Images."

Possible values

- inherit.
- repeat (default)—Tiled, repeating the image both horizontally and vertically.
- repeat-x—Repeating the image horizontally only.
- repeat-y—Repeating the image vertically only.
- no-repeat—Not repeating the image at all, showing just one instance.

Example

```
body {
    background-image: url(images/sifakabg.gif);
    background-repeat: no-repeat;
}
```

Related Properties

`background`, `background-image`

border, border-top, border-right, border-bottom, border-left

The width, style, and color of a box's border.

See Chapter 5, "Layout."

Possible values

[Combination of `border-width`, `border-style`, `border-color`]

Example

```
.this {
    border-top: 1px solid black;
}
.that {
    border: 1em dotted #fc0;
}
```

Related Properties

`border-width`, `border-style`, `border-color`, `padding`, `margin`

border-collapse

Specifies which border model should be used in a table.

See Chapter 8, "Tables."

Possible values

- inherit
- separate (default)—"Separated borders" model: Cells have their own borders.
- collapse—"Collapsing borders" model: Cells share adjacent borders. This pushes all of the cells together, leaving only the wider of the two adjacent borders visible.

Example

```
table {
    border-collapse: collapse;
}
td {
    border: 1px solid #ccc;
}
```

Related Properties

`border-spacing`

border-color, border-top-color, border-right-color, border-bottom-color, border-left-color

The color of a box's border.

See Chapter 5, "Layout."

Possible values

- inherit
- transparent
- [color]

The value for `border-color` can comprise one value (uniform border color), two values ([top/bottom][left/right]), three values ([top][left/right][bottom]), or four values ([top][right][bottom][left]).

Example

```
.flamingo {
    border-right-color: red;
}
#peach {
    border-color: #cc3421;
}
#tree {
    border-color: #fc0 blue #cf0;
}
```

Related Properties

`border`

border-spacing

Specifies the spacing between the borders of adjacent table cells in the "separated borders" model. Not supported by IE.

Possible values

- inherit
- [length]

`border-spacing` can have one value such as `5px` to specify spacing on all sides or two values such as `5px 10px` to specify the horizontal (first value) and vertical (second value) spacing.

Example

```
table {
    border-collapse: separate;
    border-spacing: 0.25em 0.5em;
}
td {
    border: 1px solid #ccc;
}
```

Related properties

`border-collapse`

border-style, border-top-style, border-right-style, border-bottom-style, border-left-style

The style of a box's border.

See Chapter 5, "Layout."

Possible values

- inherit
- none—No border.

- dotted—A series of dots (IE 6 and below will display this as dashes if the border width is one pixel).
- dashed—A series of dashes.
- solid—A solid line.
- double—Two solid lines.
- groove—Patterned border that is supposed to represent a carved groove (opposite of ridge). Renders differently in different browsers.
- ridge—Patterned border that is supposed to represent an embossed ridge (opposite of groove). Renders differently in different browsers.
- inset—Patterned border that is supposed to represent an inset depression (opposite of outset). Renders differently in different browsers.
- outset—Patterned border that is supposed to represent an outset extrusion (opposite of inset). Renders differently in different browsers.
- hidden—Used with tables. Same as none, except where there are conflicting borders. Not supported by IE.

The value for border-style can comprise one value (uniform border style), two values ([top/bottom][left/right]), three values ([top][left/right][bottom]), or four values ([top][right][bottom][left]).

Example

```
.curtains {
    border-right-style: solid;
}
.blinds {
    border-style: dotted dashed;
}
```

Related properties

border

border-width, border-top-width, border-right-width, border-bottom-width, border-left-width

The width of a box's border.

See Chapter 5, "Layout."

Possible values

- inherit
- thin
- medium
- thick
- [length]

Value for border-width can comprise one value (uniform border width), two values ([top/bottom][left/right]), three values ([top][left/right][bottom]), or four values ([top][right][bottom][left]).

Example

```
#bender {
    border-left-width: 2px;
}
#fry {
    border-width: 1px 4px 1px 100px;
}
```

Related properties

`border`

bottom

For absolutely positioned boxes, specifies how far from the bottom of the containing positioned box (or, if there isn't one, the page) the box should be.

For relatively positioned boxes, specifies how far from the bottom a box should be shifted.

See Chapter 5, "Layout."

Possible values

- inherit
- auto (default)
- [percentage]
- [length]

Example

```
#justaveit {
    position: absolute;
    bottom: 2em;
}
```

Related properties

`top`, `right`, `left`, `position`

caption-side

Specifies on which side of the table a table-caption box (such as the default style of the HTML `caption` element) will be placed. Not supported by IE.

See Chapter 8, "Tables."

Possible values

- inherit
- top (default)
- right
- bottom
- left

Example

```
caption {
    caption-side: right;
}
```

Related properties

[none]

clear

Specifies how a box is placed after a floated box.

See Chapter 5, "Layout."

Possible values

- inherit
- none (default)—Floated boxes are not cleared; content will flow around them.
- left—Clears all left-floated boxes and places the box underneath.
- right—Clears all right-floated boxes and places the box underneath.
- both—Clears all floated boxes and places the box underneath.

Example

```
#canoe { float: left; }
#fish { clear: left; }
```

Related properties

`float`

clip

Specifies the area of an absolutely positioned box that should be visible.

See Chapter 5, "Layout."

Possible values

- inherit.
- auto (default)—No clipping.
- rect([top] [right] [bottom] [left])—Clips to the shape of a rectangle defined by the four coordinates (offset from the top left corner).

Example

```
#spod {
    position: absolute;
    clip: rect(10px 50px 30px 10px);
}
```

Related properties

overflow

color

Foreground color. This applies most commonly to text, but also to borders.

See Chapter 1, "Getting Started."

Possible values

- inherit
- [color]

Example

```
body {
    font-family: "Times New Roman", Times, serif;
    color: white;
    background-color: black;
}
code {
    color: #900;
}
```

Related properties

`background-color`

content

Generated content that can be displayed before or after a box. Used in conjunction with the `:before` and `:after` pseudo-elements. Not widely supported.

Possible values

- inherit
- normal—No generated content.
- open-quote—The content defined by the quotes property (‘ “ ‘ by default).
- close-quote—The content also defined by the quotes property (‘ “ ‘ by default).
- no-open-quote—No opening quote is applied, but the nesting order is maintained.
- no-close-quote—No closing quote is applied, but the nesting order is maintained.
- attr([attribute name])—The value of attribute [attribute name] in the HTML tag that is the subject of the selector.
- counter([name], [style])—The current value of counter [name]. The optional [style] is a value equivalent to that of list-style-type.
- counters([name], [string], [style])—The current values of all counters called [name], separated by [string]. The optional [style] is a value equivalent to that of list-style-type.
- [URI]
- [string]

Example

```
p:before { content: url(images/quote.gif); }
p:after { content: close-quote; }
li:before { content: ">>"; }
```

Related properties

`:before` and `:after` pseudo-elements, `quotes`

counter-increment

Increments a named counter. Not widely supported.

Possible values

- inherit
- none (default)
- [name] [number]—The name of the counter optionally followed by the number that the counter should be increased by (default is 1). This can be a chain of names and numbers such as chapter section 2, which will increase chapter by 1 and section by 2.

Example

```
h3:before {
     content: counter(section);
     counter-increment: section;
}
```

Related properties

`counter-reset`, `content`

counter-reset

Resets a named counter. Not widely supported.

Possible values

- inherit
- none (default)
- [name] [number]—The name of the counter optionally followed by the number that the counter resets to (default is 0). This can be a chain of names and numbers such as chapter 2 section 1 subsection, which will reset chapter to 2, section to 1, and subsection to 0.

Example

```
h2:before {
    content: counter(chapter);
    counter-increment: chapter;
    counter-reset: section;
}
```

Related properties

`counter-increment`, `content`

cursor

The appearance of the cursor when it passes over a box.

Possible values

- inherit
- auto (default)—Changes depending on the situation (a pointer when the cursor is over a link; an I-beam when it is over text, etc.).
- crosshair—A thin plus-sign-like cross.
- default—The platform's default cursor; usually an arrow.
- help—Used to indicate that there is help for the element that is being hovered over; usually a question mark.

- move—Used to indicate something that should be moved; usually a four-way arrow.
- n-resize—Used to indicate something that should be scaled upwards; usually an up/down arrow.
- ne-resize—Used to indicate something that should be scaled upwards and to the right; usually a diagonal arrow.
- e-resize—Used to indicate something that should be moved to the right; usually a left/right arrow.
- se-resize—Used to indicate something that should be moved downwards and to the right; usually a diagonal arrow.
- s-resize—Used to indicate something that should be moved downwards; usually an up/down arrow.
- sw-resize—Used to indicate something that should be moved downwards and to the left; usually a diagonal arrow.
- w-resize—Used to indicate something that should be moved to the left; usually a left/right arrow.
- nw-resize—Used to indicate something that should be moved upwards and to the left; usually a diagonal arrow.
- text—Used to indicate text; usually an I-beam.
- pointer—Used to indicate a link; usually a pointing hand.
- progress—Used to indicate that the program is processing something, but that it can still be interacted with; usually an arrow coupled with a timer.
- wait—Used to indicate that the user should wait while a program is busy; usually a timer.
- [URI]—A custom-made image.

Example

```
acronym { cursor: help; }
```

Related properties

`:hover` pseudo class

direction

The writing direction and the direction of embeddings and overrides (used in conjunction with `unicode-bidi`).

Possible values

- inherit
- ltr (default)—Left to right
- rtl—Right to left

Example

```
p { direction: rtl; }
```

Related properties

`unicode-bidi`

display

The display type of a box.

See Chapter 5, "Layout."

Possible values

- inherit
- none—No display at all
- inline—An inline box
- block—A block box

- inline-block—Effectively a block box inside an inline box. IE will only apply inline-block to elements that are traditionally inline such as span or a but not p or div. Loopy.
- run-in—Either an inline or block box, depending on the context. If a block box follows the run-in box, the run-in box becomes the first inline box of that block box; otherwise, it becomes a block box itself. Crazy. Not widely supported.
- list-item—The equivalent of the default styling of the HTML li element.
- table—A block-level table; the equivalent of the default styling of the HTML table element. Not supported by IE.
- inline-table—An inline-level table. Not supported by IE.
- table-row-group—The equivalent of the default styling of the HTML tbody element. Not supported by IE.
- table-header-group—The equivalent of the default styling of the HTML thead element. Not supported by IE.
- table-footer-group—The equivalent of the default styling of the HTML tfoot element. Not supported by IE.
- table-row—The equivalent of the default styling of the HTML tr element. Not supported by IE.
- table-column-group—The equivalent of the default styling of the HTML colgroup element. Not supported by IE.
- table-column—The equivalent of the default styling of the HTML col element. Not supported by IE.
- table-cell—The equivalent of the default styling of the HTML td or th elements. Not supported by IE.
- table-caption—The equivalent of the default styling of the HTML caption element. Not supported by IE.

Example

```
.darwin { display: block; }
.lamarck { display: none; }
.linnaeus { display: table; }
```

Related properties

`visibility`

empty-cells

Whether empty table cells should be shown or not.

See Chapter 8, “Tables.”

Possible values

- inherit
- show (default)
- hide

Example

```
table { empty-cells: hide }
```

Related properties

[none]

float

Specifies whether a fixed-width box should float, shifting it to the right or left with surrounding content flowing around it.

See Chapter 5, “Layout.”

Possible values

- inherit

- left—Floats the box to the left with surrounding content flowing to the right.
- right—Floats the box to the right with surrounding content flowing to the left.
- none (default).

Example

```
#boondoggle {
    width: 20em;
    float: left;
}
```

Related properties

clear, position

font

Font characteristics combining italics, small-caps, boldness, size, line-height, and font name in one property.

See Chapter 2, "Text."

Possible values

[Combination of font-style, font-variant, font-weight, font-size, line-height, and font-family] Taking the format of *font: font-style font-variant font-weight font-size/line-height font-family*

Only the font-size and font-family parts are required.

Example

```
p {
    font: italic small-caps bold 0.8em/1.5 arial, Helvetica, sans-serif;
}
booba {
    font: bold 3.5em arial, helvetica, sans-serif;
}
```

Related properties

`font-style`, `font-variant`, `font-weight`, `font-size`, `line-height`, `font-family`

font-family

Which font you want your text to appear in.

See Chapter 2, "Text."

Possible values

- inherit
- [Font name]—Note that if a font name consists of more than one word it should be stated in quotation marks.
- [multiple font names separated by commas]—You can specify more than one font by separating them with commas. By doing this, if a browser cannot find the first choice font, it will move on to the next in the list.

Example

```
body { font-family: "Times New Roman"; }
h2 { font-family: arial, helvetica, sans-serif; }
```

Related properties

`font`, `font-size`

font-size

The size of the font.

See Chapter 2, "Text."

Possible values

- inherit
- larger

- smaller
- xx-small
- x-small
- small
- medium (default)
- large
- x-large
- xx-large
- [percentage]
- [length]

Example

```
body { font-size: 80%; }
h1 { font-size: 2em; }
```

Related properties

`font`, `font-family`

font-style

Italic and oblique characteristics of a font.

See Chapter 2, "Text."

Possible values

- inherit
- normal
- italic
- oblique

Example

```
h1, h2 { font-style: italic }
```

Related properties

`font`, `font-weight`

font-variant

Used to convert lowercase letters to small uppercase letters.

See Chapter 2, "Text."

Possible values

- inherit
- normal (default)
- small-caps

Example

```
p { font-variant: small-caps; }
```

Related properties

`font`, `text-transform`

font-weight

The boldness of a font. Values 100 to 900 are supposed to be different scales of boldness, but in practice browsers tend not to reliably differentiate between nine separate levels, which is why the value of `font-weight` tends to be simply either `normal` or `bold`.

See Chapter 2, "Text."

Possible values

- inherit

- 100, 200, 300, 400, 500, 600, 700, 800, or 900
- normal—Equivalent of 400
- bolder
- bold—Equivalent of 700
- lighter

Example

```
.chubbybaby { font-weight: bold }
```

Related properties

font, font-style

height

Specifies the height of the content area of a block box (*not* including padding, border, or margin).

See Chapter 5, "Layout."

Possible values

- inherit
- auto (default)
- [percentage]
- [length]

Example

```
#monstermunch {
    padding: 1em;
    height: 3em;
}
```

Related properties

`width`, `min-height`, `max-height`

left

For absolutely positioned boxes, specifies how far from the left of the containing positioned box (or the page, if there isn't one) the box should be.

For relatively positioned boxes, specifies how far from the left a box should be shifted.

See Chapter 5, "Layout."

Possible values

- inherit
- auto (default)
- [percentage]
- [length]

Example

```
#sold {
    position: absolute;
    left: 150px;
}
```

Related properties

`right`, `top`, `bottom`, `position`

letter-spacing

The spacing between letters.

See Chapter 2, "Text."

Possible values

- inherit
- normal (default)
- [Length]

Example

```
p { letter-spacing: 0.3em }
```

Related properties

word-spacing

line-height

The height of a line of text.

See Chapter 2, "Text."

Possible values

- inherit
- normal—Usually about 1.2 times the height of the font.
- [number]—A multiple of the font size (so, in effect, the same as a value specified in ems).
- [percentage]—A percentage of the font size.
- [length]

Example

```
p { line-height: 1.5 }
```

Related properties

font, font-size

list-style

A shorthand property used to specify the styles of a list item marker.

See Chapter 6, "Lists."

Possible values

[combination of `list-style-type`, `list-style-position`, and `list-style-image`]

Example

```
ul { list-style: none url(images/arrow.gif) inside;    }
ul ul { list-style: disc outside; }
#nav ul { list-style: none; }
```

Related properties

`list-style-type`, `list-style-position`, `list-style-image`

list-style-image

Specifies an image to be used as the list marker for a list item.

See Chapter 6, "Lists."

Possible values

- inherit
- none (default)
- [URI]

Example

```
ul { list-style-image: url(images/arrow.gif); }
```

Related properties

`list-style`, `list-style-type`

list-style-position

Specifies whether the list marker for a list item should appear inside or outside the list-item box. By default, lists will place the marker of each list item outside the content box, which means that when it comes to styling list items with backgrounds or borders, for example, the bullet will aloofly hang about outside. You can pull the marker inside the content box to deal with such circumstances by setting the `list-style-position` property to `inside`.

See Chapter 6, "Lists."

Possible values

- inherit
- outside (default)
- inside

Example

```
ul { list-style-position: inside; }
```

Related properties

`list-style`

list-style-type

The type of the list marker bullet or numbering system within a list. These can be applied to any (non-definition) lists regardless of whether they are ordered or unordered.

See Chapter 6, "Lists."

Possible values

- inherit
- none—No list marker. This can be handy when you want to present lists that don't appear in main content and don't need to stand out from the crowd with markers—as in navigation bars, for example.

- disc—Solid circle
- circle—Hollow circle
- square—Solid square
- decimal (default for ol elements)—1, 2, 3, 4, etc.
- decimal-leading-zero—01, 02, 03 ... 10, 11, etc. Not supported by IE.
- lower-roman—i, ii, iii, iv, etc.
- upper-roman—I, II, III, IV, etc.
- lower-greek—Greek characters. Not supported by IE.
- lower-latin—a, b, c, d, etc. Not supported by IE.
- upper-latin—A, B, C, D, etc. Not supported by IE.
- armenian—Armenian characters. Not widely supported.
- georgian—Georgian characters. Not widely supported.

Example

```
ol { list-style-type: lower-roman; }
ul { list-style-type: square; }
ul ul { list-style-type: circle; }
```

This example applies lower-roman numerals to ordered lists, square bullets to top-level unordered lists, and circular bullets to all unordered lists nested within unordered lists.

Related properties

list-style, **list-style-image**

margin, margin-top, margin-right, margin-bottom, margin-left

The margin of a box.

See Chapter 5, "Layout."

Possible values

- inherit
- [percentage]
- [length]
- The value for margin can comprise one value (uniform margin), two values ([top/bottom][left/right]), three values ([top][left/right][bottom]), or four values ([top][right][bottom][left]).

Example

```
#badger {
    margin-top: 3em;
}
#wolverine {
    margin: 1em 100px;
}
#pineapple {
    margin: 1em 3em 10px 0.5em;
}
```

Related properties

padding, border

max-height

The maximum height of a box. Not supported by IE 6 or below.

Possible values

- inherit
- none
- [percentage]
- [length]

Example

```
#bing { max-height: 300px; }
```

Related properties

`min-height`, `height`, `max-width`, `min-width`

max-width

The maximum width of a box. Not supported by IE 6 or below.

Possible values

- inherit
- none
- [percentage]
- [length]

Example

```
#bong { max-width: 780px; }
```

Related properties

`min-width`, `width`, `max-height`, `min-height`

min-height

The minimum height of a box. Not supported by IE 6 or below (where `height` acts the same).

Possible values

- inherit
- none
- [percentage]
- [length]

Example

```
#beng { min-height: 5em; }
```

Related properties

`max-height`, `height`, `min-width`, `max-width`

min-width

The minimum width of a box. Not supported by IE 6 or below.

Possible values

- inherit
- none
- [percentage]
- [length]

Example

```
#bung { min-width: 300px; }
```

Related properties

`max-width`, `width`, `min-height`, `max-height`

orphans

Used in paged media. The minimum number of lines in an element that must be left at the bottom of a page. Not widely supported.

Possible values

- inherit
- [number]—(default is 2)

Example

```
p { orphans: 3; }
```

Related properties

`widows`

outline

Specifies an outline for a box. Rendered around the outside of the border and on top of the box, so it does not affect its size or position. The value can combine `outline-color`, `outline-style`, and `outline-width`. Not supported by IE/Win or Mozilla at the time of writing.

Possible values

[Combines `outline-color`, `outline-style`, and `outline-width`]

Example

```
.ferrari { outline: 3px solid red; }
```

Related properties

`outline-color`, `outline-style`, `outline-width`, `border`

outline-color

The color of an outline. Not supported by IE/Win or Mozilla at the time of writing.

Possible values

- invert
- [color]

Example

```
.redbull { outline-color: blue; }
```

Related properties

`outline`, `border-color`

outline-style

The style of an outline. Not supported by IE/Win or Mozilla at the time of writing.

Possible values

- none—No border.
- dotted—A series of dots.
- dashed—A series of dashes.
- solid—A solid line.
- double—Two solid lines.
- groove—Patterned border that is supposed to represent a carved groove (opposite of ridge).
- ridge—Patterned border that is supposed to represent an embossed ridge (opposite of groove).
- inset—Patterned border that is supposed to represent an inset depression (opposite of outset).
- outset—Patterned border that is supposed to represent an outset extrusion (opposite of inset).
- hidden—Used with tables. Same as none, except where there are conflicting borders.

Example

```
.honda { outline-style: solid; }
```

Related properties

`outline`, `border-style`

outline-width

The width of an outline. Not supported by IE/Win or Mozilla at the time of writing.

Possible values

- thin
- medium
- thick
- [length]

Example

```
.williams { outline-width: 0.5em; }
```

Related properties

`outline`, `border-width`

overflow

Specifies what should happen to the overflow—the portions of content that do not fit inside the box.

See Chapter 5, "Layout."

Possible values

- inherit
- visible (default)—Overflow spills over the box.
- hidden—Any content that doesn't fit in the box will be "clipped"—cut off at the edge of the box.
- scroll—Scrollbars appear, allowing the user to scroll the box to see the overflow.
- auto—Scrollbars will only be displayed if they are necessary (whereas overflow: scroll will show them even if the content of the box fits without any overflow).

Example

```
#content {
     width: 500px;
     height: 4em;
     overflow: hidden;
}
```

Related properties

`clip`, `height`, `width`

padding, padding-top, padding-right, padding-bottom, padding-left

The padding of a box.

See Chapter 5, "Layout."

Possible values

- inherit
- [percentage]
- [length]

Value for padding can comprise one value (uniform padding), two values ([top/bottom][left/right]), three values ([top][left/right][bottom]), or four values ([top][right][bottom][left]).

Example

```
#flump { padding: 10em 2em; }
```

Related properties

`border`, `margin`

page-break-after

Used in paged media. How a page break should be applied after a block box, forcing a new page box. Not widely supported.

Possible values

- inherit
- auto (default)—Does not force or forbid a page break.
- always—Always forces a page break.
- avoid—Forbids a page break.
- left—Forces either one or two page breaks so that the next page is a left page.
- right—Forces either one or two page breaks so that the next page is a right page.

Example

```
#europe { page-break-after: left; }
```

Related properties

page-break-before, page-break-inside

page-break-before

Used in paged media. How a page break should be applied before a block box, forcing a new page box. Not widely supported.

Possible values

- inherit
- auto (default)—Does not force or forbid a page break.
- always—Always forces a page break.

- avoid—Forbids a page break.
- left—Forces either one or two page breaks so that the next page is a left page.
- right—Forces either one or two page breaks so that the next page is a right page.

Example

```
#antarctica { page-break-before: always; }
```

Related properties

page-break-after, page-break-inside

page-break-inside

Used in paged media. How a page break should be applied inside a block box, forcing a new page box. Not widely supported.

Possible values

- inherit
- auto (default)—Does not force or forbid a page break.
- avoid—Forbids a page break.

Example

```
#africa { page-break-inside: avoid; }
```

Related properties

page-break-after, page-break-before

position

How a box should be positioned.

See Chapter 5, “Layout.”

Possible values

- inherit
- static (default)—Follows the normal flow.
- relative—Relative position that is offset from the initial normal position in the flow.
- absolute—Taken out of the flow and offset according to the containing block.
- fixed—The same as absolute only the fixed box will remain fixed in the viewport and not scroll (or will appear on every printed page). Not supported by IE 6 or below.

Example

```
#oogabooga {
    position: relative;
    left: 1em;
    top: 1em;
}
```

Related properties

`float`, `top`, `bottom`, `left`, `right`

quotes

What form the quotes of the `open-quote` and `close-quote` values of the `content` property should take. Not supported by IE.

Possible values

- inherit
- none
- [string] [string]—The first string is that used for the open-quote value and second string for the close-quote value.

- [string] pairs can be repeated, whereby each consecutive pair will represent the quotes for the next level of embedding. For example, ‘“‘ ‘“‘ “‘“ “‘“ will specify that quotes within a quoted element will be surrounded by ‘ characters. Not widely supported.

Example

```
q { quotes: '"' '"' }
```

Related properties

`content`

right

For absolutely positioned boxes, specifies how far from the right of the containing positioned box (or the page, if there isn't one) the box should be.

For relatively positioned boxes, specifies how far from the right a box should be shifted.

See Chapter 5, “Layout.”

Possible values

- inherit
- auto (default)
- [percentage]
- [length]

Example

```
#tolet {
    position: relative;
    right: 2em;
}
```

Related properties

`left`, `top`, `bottom`, `position`

table-layout

Used to specify the algorithm that should be used to render a fixed-width table. Not supported by early versions of IE.

See Chapter 8, "Tables."

Possible values

- inherit
- auto (default)—Column widths are determined by the cells in all rows.
- fixed—Column widths are determined by the cells in the first row only. Table renders faster.

Example

```
table {
     table-layout: fixed;
     width: 100%;
}
```

Related properties

`width`

text-align

Horizontally aligns text within a block box, such as a default paragraph.

See Chapter 2, "Text."

Possible values

- inherit
- left

- right
- center
- justify

Example

```
p { text-align: right; }
```

Related properties

[none]

text-decoration

Underline, over-line, or strikethrough.

See Chapter 2, "Text."

Possible values

- inherit
- none
- underline—Line underneath text.
- overline—Line above text.
- line-through—Line through the middle of text.
- blink—Not supported by IE, or used by sensible people.

Example

```
ins { text-decoration: none }
```

Related properties

border

text-indent

The indentation of the first line of text in a block box.

See Chapter 2, "Text."

Possible values

- inherit
- [Percentage]
- [Length]

Example

```
p { text-indent: 1em }
```

Related properties

[none]

text-transform

Converts the case of letters.

See Chapter 2, "Text."

Possible values

- inherit
- none (default)
- capitalize—Capitalizes the first letter of every word.
- uppercase—Forces every letter into uppercase.
- lowercase—Forces every letter into lowercase.

Example

```
h1, h2 { text-transform: uppercase }
```

Related properties

`font-variant`

top

For absolutely positioned boxes, specifies how far from the top of the containing positioned box (or the page, if there isn't one) the box should be.

For relatively positioned boxes, specifies how far from the top a box should be shifted.

See Chapter 5, "Layout."

Possible values

- inherit
- auto (default)
- [percentage]
- [length]

Example

```
#forsale {
     position: absolute;
     top: 25%;
}
```

Related properties

`bottom`, `left`, `right`, `position`

unicode-bidi

Used in conjunction with direction, specifies how text is mapped to the Unicode algorithm, determining its directionality.

Possible values

- inherit
- normal (default)—No additional embedding. Applies the implicit Unicode character order.
- embed—Opens an additional level of embedding within the algorithm whilst maintaining the implicit Unicode character order.
- bidi-override—Opens an additional level of embedding and overrides the Unicode character ordering, reordering the sequence to the value of the direction property.

Example

```
.hebrew {
    direction: rtl;
    unicode-bidi: bidi-override;
}
```

Related properties

`direction`

vertical-align

The vertical position of an inline box, or content within a table cell. Values such as **`top`**, **`middle`**, **`bottom`**, **`text-top`** and **`text-bottom`** rely on the styled box being smaller than some or all of the text in the rest of the line (otherwise it will already be at all of those positions).

See Chapter 2, "Text".

Possible values

- inherit
- [length]—Raises (positive value) or lowers (negative value) the box; 0 is equal to the baseline.

- [percentage]—Raises (positive value) or lowers (negative value) the box with regard the value of line-height; 0% is equal to the baseline, 100% is one times the line-height, etc.
- baseline (default)—Aligns the baseline of a box with the baseline of its parent box.
- sub—Lowers the baseline to subscript level.
- super—Raises the baseline to superscript level.
- top—Aligns to the top of the line.
- text-top—Aligns to the top of the font of the parent box.
- middle—Aligns to the middle of the font of the parent box.
- bottom—Aligns to the bottom of the line.
- text-bottom—Aligns to the bottom of the font of the parent box.

Example

```
.power {
    font-size: 80%;
    vertical-align: super;
}
```

Related properties

line-height

visibility

Makes a box is visible or invisible.

See Chapter 5, "Layout."

Possible values

- inherit
- visible (default)

- hidden—Nothing will be visible, but unlike display: none, the box and its dimensions will still affect layout.
- collapse—Same as hidden except when applied to rows, row groups, columns, or column group boxes, when it should have the same visual representation of display: none whilst maintaining the cell heights or widths that will affect row heights and column widths. At the time of writing, those browsers that "support" this value (IE does not) actually render collapse the same as hidden no matter what the situation.

Example

```
p.flummox { visibility: hidden; }
```

Related properties

display

white-space

How the white space (such as new lines or a sequence of spaces) inside a box should be handled.

See Chapter 2, "Text."

Possible values

- inherit
- normal—White space is collapsed and lines are broken to fit.
- pre—White space is maintained and lines are not broken. The equivalent of the default styling of the HTML pre element.
- nowrap—White space is collapsed but lines are not broken.
- pre-wrap—White space is maintained but lines are broken. Not recognized by IE.
- pre-line—White space is collapsed except for new lines, which are not. Lines are also broken to fit. Not recognized by IE.

Example

```
pre { white-space: normal; }
#pow { white-space: pre; }
```

Related properties

[none]

widows

Used in paged media. The minimum number of lines in a box that must be left at the top of a page. Not widely supported.

Possible values

- inherit
- [number]—(default is 2)

Example

```
p { widows: 4; }
```

Related properties

orphans

width

The width of the content area of a block box (*not* including padding, border, or margin).

See Chapter 5, "Layout."

Possible values

- inherit
- auto (default)

- [percentage]
- [length]

Example

```
#jelly { width: 212px; }
```

Related properties

`height`, `min-width`, `max-width`

word-spacing

The spacing between words.

See Chapter 2, "Text."

Possible values

- inherit
- normal (default)
- [Length]

Example

```
p {
     letter-spacing: 0.3em;
     word-spacing: 1em;
}
```

Related properties

`letter-spacing`

z-index

The order of positioned boxes in the z-axis. The higher the number, the higher that box will be in the stack (so if, for example, one box overlaps another, the box with the higher `z-index` will be on top of the other box).

See Chapter 5, "Layout."

Possible values

- inherit
- auto (default)
- [number]

Example

```
div { position: absolute; }
#kidkoala { z-index: 2 }
#mrscruff { z-index: 1 }
```

Related properties

`position`

Index

A

<a> tag, 62–63, 209–210
<abbr></abbr> tag, 210
abbreviations, structuring text, 43–44
absolute positioning, 108–109
absolute units, 26
    layout, 113–114
    styling text, 51–52
access keys
    link accessibility, 68
    major problems, 68–70
accessibility
    forms, 186–187
    links
        access keys, 68–70
        adjacent links, 71–72
        pop-ups, 71
        skipping navigation, 72–74
        tabbing, 67–68
        titles, 70–71
    tables
        cell to header association, 165–167
        header to cell association, 165
        summaries, 164
<acronym></acronym> tag, 210–211
acronyms, structuring text, 43–44
:active pseudo-class, 265–266
active states, 67
<address></address> tag, 47, 211
addresses, structuring text, 47
adjacent links, accessibility, 71–72
adjacent sibling selectors, 24
:after pseudo-elements, 269
algorithms, fixed layout, 169–170
alignment, text styling
    horizontal, 58
    vertical, 59
ancestors, 3
anchor elements, links, 62–64
<area /> tag, 212–213
at-rules
    @import, 270
    @media, 270–271
    @page, 271
    selectors, 23
attributes
    <a> tag, 209–210
    <abbr></abbr> tag, 210
    <acronym></acronym> tag, 211
    <address /></address/> tag, 211
    <area /> tag, 212
    bad, 264
    <base /> tag, 213
    <bdo></bdo> tag, 214
    <blockquote></blockquote> tag, 214–215
    <body></body> tag, 215

 tag, 216
    <button></button> tag, 216–217
    <caption></caption> tag, 217
    <cite></cite> tag, 218
    <code></code> tag, 218–219
    <col /> tag, 219–220
    <colgroup></colgroup> tag, 220
    <dd></dd> tag, 221
    <del></del> tag, 222
    <dfn></dfn> tag, 222–223
     tag, 223
    <dl></dl> tag, 224
    <dt></dt> tag, 224–225
     tag, 225
    <fieldset></fieldset> tag, 225–226
    <form></form> tag, 226
    formats, 207–208
    <h1></h1> tag, 227–228

attributes *(continued)*
<h2></h2> tag, 227–228
<h3></h3> tag, 227–228
<h4></h4> tag, 227–228
<h5></h5> tag, 227–228
<h6></h6> tag, 227–228
<head></head> tag, 228
HTML, 2–4
core, 4–7
il8n, 7
<html></html> tag, 229
<img /> tag, 230
<input /> tag, 231–232
<ins></ins> tag, 232–233
<kbd></kbd> tag, 233
<label></label> tag, 233–234
<legend></legend> tag, 234
 tag, 234–235
<link> tag, 235–236
<map></map> tag, 236
<meta /> tag, 237
<noscript></noscript> tag, 238
<object></object> tag, 239–240
 tag, 240
<optgroup></optgroup> tag, 241
<option></option> tag, 241–242
 tag, 242–243
<param /> tag, 243–244
<pre></pre> tag, 244
<q></q> tag, 244–245
<samp></samp> tag, 245
<script></script> tag, 246
<select></select> tag, 247
selectors, 24
 tag, 248
 tag, 248–249
<style></style> tag, 249–250
<table></table> tag, 250–251
<tbody></tbody> tag, 251–252
<td></td> tag, 252–254
<textarea></textarea> tag, 254–255
<tfoot></tfoot> tag, 255–256
<th></th> tag, 256–258
<thead></thead> tag, 258–259
<title></title> tag, 259–260
<tr></tr> tag, 260–261
 tag, 261–262
author style sheets, 28

B

<b> tag, 40
background-attachment property, 83, 272–273
background-color property, 50, 85, 273–274
background-image property, 82–87, 274
background images, 82–88
background-position property, 83, 275
background property, 272
background-repeat property, 83, 275–276
backgrounds, styling form fields, 189–190
bad tags, 262–264
<base /> tag, 14–15, 213
base text colors, 50
<bdo></bdo> tag, 47, 214
:before pseudo-element, 269
bidirectional text, structuring text, 47
block elements, 4, 175–182, 208
block value, 105
<blockquote></blockquote> tag, 42, 214–215
<body></body> tag, 215–216
body element, 8, 12–16
bold, styling text, 54–55
border-bottom-color property, 278
border-bottom property, 276–277
border-bottom-style property, 279–280
border-bottom-width property, 281
border-collapse property, 277
border-color property, 278
border-left-color property, 278
border-left property, 276–277
border-left-style property, 279–280
border-left-width property, 281
border property, 167–169, 188–189, 276–277
border-right-color property, 278
border-right property, 276–277
border-right-style property, 279–280
border-right-width property, 281
border-spacing property, 278–279
border-style property, 279–280
border-top-color property, 278
border-top property, 276–277
border-top-style property, 279–280

border-top-width property, 281
border-width property, 281
borders
    box model layout, 98–100
    collapsing tables, 167–169
    images, img element, 77
    styling form fields, 188–189
bottom property, 281–282
Box Model Hack, 102–103
box model layout, 94
    borders, 98–100
    Box Model Hack, 102–103
    margins, 100–103
    padding, 97–98
    width and height, 95–97

 tag, 216
browsers
    displaying fonts, 49
    style sheets, 28–29
bullets, lists, 142–144
button attribute, input element, 182
<button></button> tag, 216–217

C

<caption></caption> tag, 217–218
caption-side property, 161, 282–283
captions, tables, 160–161
Cascading Style Sheets. *See* CSS
cells, tables
    cell to header association, 165–167
    empty, 170
    header to cell association, 165
    merging, 158–160
checkbox attribute, input element, 177–178
child
    nested elements, 3
    selectors, 24
cite attribute, structuring text, 42
<cite></cite> tag, 218
    structuring text, 42
class attribute, 4–7, 19
    tag application, 207
class selectors, 18–20
classid attribute, 152
clear property, 283
client-side image maps, 82
clip property, 97, 283–284
<code></code> tag, 44–46, 218–219
codebase attribute, 152
<col /> tag, 219–220
<colgroup></colgroup> tag, 220–221
collapsing margins, 100–102
color
    styling text, 50
    values, 27–28
color property, 50, 284–285
colspan attribute, 158–160
columns
    page layouts, 120–122
        floating, 123
        multiple, 124–126
        solid navigation, 122–123
    tables, targeting, 162–164
comments
    CSS, 33
    HTML, 13
computer code, structuring text, 44–46
content
    <a> tag, 209–210
    <abbr></abbr> tag, 210
    <acronym></acronym> tag, 211
    <address></address> tag, 211
    <area /> tag, 212
    <base /> tag, 213
    <bdo></bdo> tag, 214
    <blockquote></blockquote> tag, 214–215
    <body></body> tag, 215

 tag, 216
    <button></button> tag, 217
    <caption></caption> tag, 217–218
    <cite></cite> tag, 218
    <code></code> tag, 218–219
    <col /> tag, 219–220
    <colgroup></colgroup> tag, 220
    <dd></dd> tag, 221
    <del></del> tag, 222
    <dfn></dfn> tag, 222–223
     tag, 223
    <dl></dl> tag, 224
    <dt></dt> tag, 224–225
    elements, 208
     tag, 225

content *(continued)*
<fieldset></fieldset> tag, 225–226
<form></form> tag, 227
<h1></h1> tag, 227–228
<h2></h2> tag, 227–228
<h3></h3> tag, 227–228
<h4></h4> tag, 227–228
<h5></h5> tag, 227–228
<h6></h6> tag, 227–228
<head></head> tag, 228
HTML, xix–xxiii
<html></html> tag, 229
<img /> tag, 230
<input /> tag, 231–232
<ins></ins> tag, 232–233
<kbd></kbd> tag, 233
<label></label> tag, 233–234
<legend></legend> tag, 234
 tag, 234–235
<link> tag, 235–236
<map></map> tag, 236
<meta /> tag, 237
<noscript></noscript> tag, 238
<object></object> tag, 239–240
 tag, 240
<optgroup></optgroup> tag, 241
<option></option> tag, 241–242
 tag, 242–243
<param /> tag, 243–244
<pre></pre> tag, 244
<q></q> tag, 244–245
<samp></samp> tag, 245
<script></script> tag, 246
<select></select> tag, 247
 tag, 248
 tag, 248–249
<style></style> tag, 249–250
<table></table> tag, 250–251
<tbody></tbody> tag, 251–252
<td></td> tag, 252–254
<textarea></textarea> tag, 254–255
<tfoot></tfoot> tag, 255–256
<th></th> tag, 256–258
<thead></thead> tag, 258–259
<title></title> tag, 259–260
<tr></tr> tag, 260–261
types
HTML document declaration, 11–12
server-side scripting language, 12
 tag, 261–262
content property, 285–286
core attributes, 4–7, 207–208
Cork'd website, 172
counter-increment property, 286
counter-reset property, 286–287
CSS (Cascading Style Sheets), xix–xxiii, xvii.
See also style sheets
applying to HTML
embedded CSS, 32–33
external CSS, 34
inline CSS, 32
at-rules, 23, 270–271. *See also* at-rules
comments, 33
images
background, 82–87
decorative effects, 86–87
text graphical alternatives, 88–92
multiple media specific styles, 195–196
media attribute, 196–203
style sheet application, 204–205
page layouts, 119–120
creating columns, 120–126
footers, 127–130
headers, 126–127
properties, 23–25, 271–322. *See also* properties
pseudo-classes, 20, 265–269. *See also* pseudo-classes
pseudo-elements, 20, 269–270. *See also* pseudo-elements
rules, 17
selectors, 18–23
values, 25–31
CSS Zen Garden website, 89, 118
cursor property, 287–288

D

datetime attribute, 46
<dd></dd> tag, 221
declarations, HTML document structure, 8–12
definition lists, 138–139
<del></del> tag, 46, 222

deletions, structuring text, 46–47
descendants, 3
<dfn></dfn> tag, 222–223
Digital Web Magazine website, 135
dir attribute, 47
direction property, 289
display property, 104–107, 289–291
 tag, 16–17, 223
<dl></dl> tag, 138–139, 224
DOCTYPE statements, 9
Document Object Model (DOM), JavaScript
    event attributes, 148–149
    manipulating, 149–150
    script element, 147–148
document structures, HTML declarations, 8–12
document types, HTML structure, 9
DOM (Document Object Model), JavaScript
    event attributes, 148–149
    manipulating, 149–150
    script element, 147–148
DOM Scripting, 150
<dt></dt> tag, 224–225

E

editing HTML
    default styles, 47
    insertions and deletions, 46–47
elastic layouts, 117–119
element attribute, 173–174
elements
    content, 208
    forms, 173–174
    HTML, 2–4
 tag, 225
embedded CSS, applying CSS to HTML, 32–33
emphasis, structuring text, 39–40
empty cells, styling tables, 170
empty-cells property, 170, 291
event attributes, 8
    general tag application, 208
    JavaScript, 148–149
eXtensible HTML (XHTML), xvii. *See also* HTML
external CSS, applying CSS to HTML, 34

F

Fahrner Image Replacement (FIR), 90
family connections, nested elements, 3
<fieldset></fieldset> tag, 185–186, 225–226
fieldsets, forms, 185–186
file attribute, input element, 181–182
FIR (Fahrner Image Replacement), 90
:first-child pseudo-class, 266
:first-letter pseudo-element, 269–270
:first-line pseudo-element, 270
:first pseudo-class, 266
fixed layout algorithms, styling tables, 169–170
fixed layouts, 113–114
fixed positioning, 110
Flash Satay, 152
float property, 110–113, 291–292
floating columns, 123
:focus pseudo-class, 266–267
focus states, 66–67
font-family property, 48–49, 293
font property, 292–293
font-size property, 293–294
font-style property, 55, 294–295
font-variant property, 295
font-weight property, 55, 295–296
fonts
    browser display, 49
    shorthand properties, 55–56
    styling form fields, 189
    styling text, 48–49
footers, page layouts, 127–130
<form></form> tag, 226–227
formats
    attributes, 207–208
    images, 81
forms, 171–172
    accessibility, 186–187
    elements, 173–174
    fields
        input element, 174–182
        select element, 183–185
        textarea element, 182–183
    fieldsets, 185–186

forms *(continued)*
styling fields, 187–188
backgrounds, 189–190
borders, 188–189
fonts, 189

G

GIF, image formats, 81
graphics. *See also* images
images text replacement, 88–92
grouped selectors, 20

H

`<h1></h1>` tag, 40–41, 227–228
`<h2></h2>` tag, 40–41, 227–228
`<h3></h3>` tag, 40–41, 227–228
`<h4></h4>` tag, 40–41, 227–228
`<h5></h5>` tag, 40–41, 227–228
`<h6></h6>` tag, 40–41, 227–228
`head` element, 8, 12–16
`<head></head>` tag, 228–229
headers
page layouts, 126–127
tables
cell to header association, 165–167
header to cell association, 165
headings, structuring text, 40–41
`height` attribute, `img` element, 80
`height` property, 95–97, 296–297
hex values, 27–28
Hicks, John, blog, 114
`hidden` attribute, `input` element, 180
horizontal alignment, styling text, 58
horizontal lists, 146
`:hover` pseudo-class, 267
hover states, 66
`href` attribute, links, 62–63
HTML, 1
applying CSS to
embedded CSS, 32–33
external CSS, 34
inline CSS, 32
attributes, 2–4
core, 4–7
il8n, 7
comments, 13
content. *See* content
document structure
`body` element, 12–16
declarations, 8–12
`div` tag, 16–17
`head` element, 12–16
JavaScript. *See* JavaScript
`span` tag, 16–17
elements, 2–4
event attributes, 8
selectors, 18
tags, 2–4
HTML Dog, 5, xv–xvi
`html` element, 8
`<html></html>` tag, 229
HTTP headers
content types, 11–12
HTML document language declaration, 11
hypertext references, links, 62–64

I

`<i>` tag, 40
`id` attribute, 4–7, 19, 207
id selectors, 18
`il8n` attribute, 7, 208
`image` attribute, `input` element, 180–181
images, 75
background, 82–87
decorative effects, 86–88
file formats, 81
`img` element, 77–79
lists, 142–144
maps, 81–82
text graphical alternatives, 88–92
`<img />` tag, 230
`img` element, 77–80
`@import` at-rule, 270
indenting, styling text, 58
inheritance, properties, 25
inline CSS, applying CSS to HTML, 32
inline elements, 4, 208
`inline` value, 105
`<input />` tag, 231–232
`input` element, 175–176
`button` attribute, 182

checkbox attribute, 177–178
file attribute, 181–182
form fields
block elements, 175–182
name attribute, 174–175
select element, 183–184
textarea element, 182–183
hidden attribute, 180
image attribute, 180–181
password attribute, 177
radio attribute, 178–179
reset attribute, 179
submit attribute, 179
text attribute, 176–177
<ins></ins> tag, 46, 232–233
insertions, structuring text, 46–47
Internet Explorer
box model layout, Box Model Hack, 102–103
unwanted lists spaces, 145
italics, styling text, 54–55

J

JavaScript
event attributes, 148–149
manipulating DOM (Document Object Model), 149–150
script element, 147–148
JPEG, image formats, 81

K

<kbd></kbd> tag, 233
Keith, Jeremy, *DOM Scripting,* 150

L

label element, 186–187
<label></label> tag, 233–234
:lang pseudo-class, 267
languages, HTML document declaration, 11
layouts, 93
application, 130–133
box model, 94
borders, 98–100
Box Model Hack, 102–103
margins, 100–103
padding, 97–98
width and height, 95–97
display property, 104–107
elastic, 117–119
fixed, 113–114
float property, 110–113
liquid, 115–116
positioning
absolute, 108–109
fixed, 110
relative, 108
static, 107
sample pages, 119–120
creating columns, 120–126
footers, 127–130
headers, 126–127
left property, 297
:left pseudo-class, 267–268
<legend></legend> tag, 234
lengths, CSS units, 25
letter-spacing property, 57, 297–298
 tag, 234–235
line breaks, structuring text, 39–40
line height property, styling text, 53–54
line-height property, 298
line heights, styling text, 53
<link> tag, 14–15, 235–236
links, 62–63
accessibility
access keys, 68–70
adjacent links, 71–72
pop-ups, 71
skipping navigation, 72–74
tabbing, 67–68
titles, 70–71
anchor elements, 62–64
hypertext references, 62–64
img element, 77
states, 65–67
URLs (Universal Resource Locator), 62
liquid layouts, 115–116
list-style-image property, 299
list-style-position property, 300
list-style property, 299
list-style-type property, 300–301

lists, 135
Internet Explorer unwanted spaces, 145
margins, 144
padding, 144
presentation structure
horizontal lists, 146
markers, 142–144
structuring
definition, 138–139
navigation, 140–142
ordered, 136–138
unordered, 136–138
longdesc attribute, img element, 77
lower case text, styling, 55

M

map element, 82
<map></map> tag, 236–237
margin-bottom property, 100, 301–302
margin-left property, 100, 301–302
margin property, 100, 301–302
margin-right property, 100, 301–302
margin-top property, 100, 301–302
margins
box model layout, 100–103
lists, 144
markers, lists, 142–144
max-height property, 96, 302–303
max-width property, 96, 303
media, 191
CSS specific styles, 195–196
media attribute, 196–203
style sheet application, 204–205
mobile devices, 192–193
printing, 193–195
screen readers, 192
@media 2006 website, 38
@media at-rule, 270–271
media attribute
CSS specific styles, 196–203
multiple media application, 204–205
merging cells, tables, 158–160
<meta /> tag, 12, 14–16, 237–238
min-height property, 96, 303–304
min-width property, 96, 304
mobile devices, multiple media, 192–193
multilanguage text, structuring text, 47
multiple columns, page layouts, 124–126
multiple media, 191
CSS specific styles, 195–196
media attribute, 196–203
style sheet application, 204–205
mobile devices, 192–193
printing, 193–195
screen readers, 192
multiple style sheets, 35

N

name attribute, 82, 174–175
navigation
link accessibility, 72–74
structuring lists, 140–142
nesting
elements, 3
lists, 137–138
selectors, 20–21
Nguyen, Michael, 138
<noscript></noscript> tag, 238
numbers
CSS units, 25
lists, 142–144

O

object element, embedding objects, 150–151
<object></object> tag, 150–151, 239–240
objects
embedding
object element, 150–151
Web standards, 151–153
JavaScript
event attributes, 148–149
manipulating, 149–150
script element, 147–148
 tag, 240
onclick attribute, 8
ondblclick attribute, 8
onkeydown attribute, 8
onkeypress attribute, 8
onkeyup attribute, 8
onmousedown attribute, 8
onmousemove attribute, 8
onmouseout attribute, 8

onmouseover attribute, 8
onmouseup attribute, 8
<optgroup></optgroup> tag, 241
<option></option> tag, 241–242
ordered lists, 136–138
orphans property, 304–305
outline-color property, 305–306
outline property, 305
outline-style property, 306
outline-width property, 307
overflow, box model layout, 96–97
overflow property, 96–97, 307–308

P

 tag, 242–243
padding
    box model layout, 97–98
    lists, 144
    shorthand values, 98
padding-bottom property, 97, 308
padding-left property, 97, 308
padding property, 308
padding-right property, 97, 308
padding-top property, 97, 308
page anchors, 63–64
@page at-rule, 271
page-break-after property, 309
page-break-before property, 309–310
page-break-inside property, 310–311
page layouts, 119–120
    creating columns, 120–122
        floating, 123
        multiple, 124–126
        solid navigation, 122–123
    footers, 127–130
    headers, 126–127
paragraphs, structuring text, 39–40
<param /> tag, 243–244
parent, nested elements, 3
password attribute, input element, 177
percentages, CSS units, 25
PNG, image formats, 81
pop-ups, link accessibility, 71
positioning, layout
    absolute, 108–109
    fixed, 110
    relative, 108
    static, 107
pre element, structuring text, 45–46
<pre></pre> tag, 244
preformatted text, structuring text, 44–46
printing
    media attribute, 196–203
    multiple media, 193–195
    printer friend versions, 196
properties
    background, 272
    background-attachment, 272–273
    background-color, 273–274
    background-image, 274
    background-position, 275
    background-repeat, 275–276
    border, 276–277
    border-bottom, 276–277
    border-bottom-color, 278
    border-bottom-style, 279–280
    border-bottom-width, 281
    border-collapse, 277
    border-color, 278
    border-left, 276–277
    border-left-color, 278
    border-left-style, 279–280
    border-left-width, 281
    border-right, 276–277
    border-right-color, 278
    border-right-style, 279–280
    border-right-width, 281
    border-spacing, 278–279
    border-style, 279–280
    border-top, 276–277
    border-top-color, 278
    border-top-style, 279–280
    border-top-width, 281
    border-width, 281
    bottom, 281–282
    caption-side, 282–283
    clear, 283
    clip, 283–284
    color, 284–285
    content, 285–286
    counter-increment, 286
    counter-reset, 286–287

properties *(continued)*
CSS, 23–25
cursor, 287–288
direction, 289
display, 289–291
empty-cells, 291
float, 291–292
font, 292–293
font-family, 293
font-size, 293–294
font-style, 294–295
font-variant, 295
font-weight, 295–296
height, 296–297
inheritance, 25
left, 297
letter-spacing, 297–298
line-height, 298
list-style, 299
list-style-image, 299
list-style-position, 300
list-style-type, 300–301
margin, 301–302
margin-bottom, 301–302
margin-left, 301–302
margin-right, 301–302
margin-top, 301–302
max-height, 302–303
max-width, 303
min-height, 303–304
min-width, 304
orphans, 304–305
outline, 305
outline-color, 305–306
outline-style, 306
outline-width, 307
overflow, 307–308
padding, 308
padding-bottom, 308
padding-left, 308
padding-right, 308
padding-top, 308
page-break-after, 309
page-break-before, 309–310
page-break-inside, 310–311
quotes, 311–312
right, 312–313
table-layout, 313
text-align, 313–314
text-decoration, 314
text-indent, 315
text-transform, 315–316
top, 316
unicode-bidi, 316–317
vertical-align, 317–318
visibility, 318–319
white-space, 319–320
widows, 320
width, 95–97, 320–321
word-spacing, 321
z-index, 322
pseudo-classes
:active, 265–266
:first, 266
:first-child, 266
:focus, 266–267
:hover, 267
:lang, 267
:left, 267–268
:right, 268
selectors, 20
:visited, 268–269
pseudo-elements
:after, 269
:before, 269
:first-letter, 269–270
:first-line, 270
selectors, 20

Q

<q></q> tag, 244–245
quotations
attribute values, 3
structuring text, 42–43
quotes property, 311–312

R

radio attribute, input element, 178–179
relative positioning, 108
relative units, 26
layout, 113–114
styling text, 51–52

reset attribute, input element, 179
RGB values, 27–28
right property, 312–313
:right pseudo-class, 268
rounded corners, background images, 86–88
row grouping, tables, 161–162
rowspan attribute, 158–160
rules, CSS, 17

S

<samp></samp> tag, 44–46, 245
sample page layouts, 119–120
    creating columns, 120–122
        floating, 123
        multiple, 124–126
        solid navigation, 122–123
    footers, 127–130
    headers, 126–127
screen readers, multiple media, 192
script element, JavaScript, 147–148
<script></script> tag, 246
scripting languages, JavaScript
    event attributes, 148–149
    manipulating DOM (Document Object Model), 149–150
    script element, 147–148
scripts, JavaScript
    event attributes, 148–149
    manipulating DOM (Document Object Model), 149–150
    script element, 147–148
select element, form fields, 183–185
<select></select> tag, 247
selectors
    CSS, 18–23
    specificity, 22–23
    versatility, 24
separated-borders model, 168
server-side image maps, 82
server-side scripting languages,
    content types, 12
shorthand properties, fonts, 55–56
siblings, 3
sizes, styling text, 50
Skip navigation links, 72–74
solid navigation columns, 122–123
spacing
    Internet Explorer lists, 145
    styling text, 57
span element, image text graphical alternatives, 88–92
 tag, 16–17, 248
specificity, selectors, 22–23
src attribute, img element, 77
static footers, 129
static positioning, 107
strikethroughs, styling text, 56–57
 tag, 248–249
structuring lists
    definition, 138–139
    navigation, 140–142
    ordered, 136–138
    unordered, 136–138
structuring text, 37–39
    abbreviations, 43–44
    acronyms, 43–44
    addresses, 47
    bidirectional text, 47
    editorial insertions and deletions, 46–47
    emphasis, 39–40
    headings, 40–41
    line breaks, 39–40
    multilanguage text, 47
    paragraphs, 39–40
    preformatted text, 44–46
    quotations, 42–43
style attribute, 4–7, 207–208
style sheets. *See also* CSS
    multiple, 35
    multiple media application, 204–205
    types, 28–31
<style></style> tag, 249–250
styling
    form fields, 187–188
        backgrounds, 189–190
        borders, 188–189
        fonts, 189
    tables
        border collapsing, 167–169
        empty cells, 170
        fixed layout algorithm, 169–170

styling *(continued)*
text
bold, 54–55
color, 50
fonts, 48–49
horizontal alignment, 58
indenting, 58
italics, 54–55
line height, 53–54
shorthand properties, 55–56
size, 50
spacing, 57
strikethroughs, 56–57
techniques, 60
underlines, 56–57
upper and lower case, 55
vertical alignment, 59
`submit` attribute, `input` element, 179
summaries, table accessibility, 164
`summary` attribute, 164

T

`table` element, 156–158
`table-layout` property, 313
`<table></table>` tag, 250–251
tables, 155–156
accessibility
cell to header association, 165–167
header to cell association, 165
summaries, 164
captions, 160–161
creating basics, 156–158
grouping rows, 161–162
merging cells, 158–160
styling
border collapsing, 167–169
empty cells, 170
fixed layout algorithm, 169–170
targeting columns, 162–164
tabs
horizontal lists, 146
link accessibility, 67–68
tags
`<a></a>`, 209–210
`<abbr></abbr>`, 210
`<acronym></acronym>`, 210–211
`<address></address>`, 47, 211
`<area />`, 212–213
attribute formats, 207–208
bad, 262–264
`<base />`, 213
`<bdo></bdo>`, 214
`<blockquote></blockquote>`, 214–215
`<body></body>`, 215–216
`<br />`, 216
`<button></button>`, 216–217
`<caption></caption>`, 217–218
`<cite></cite>`, 218
`<code></code>`, 218–219
`<col />`, 219–220
`<colgroup></colgroup>`, 220–221
`<dd></dd>`, 221
`<del></del>`, 222
`<dfn></dfn>`, 222–223
`<div></div>`, 223
`<dl></dl>`, 138–139, 224
`<dt></dt>`, 224–225
`<em></em>`, 225
`<fieldset></fieldset>`, 225–226
`<form></form>`, 226–227
`<h1></h1>`, 227–228
`<h2></h2>`, 227–228
`<h3></h3>`, 227–228
`<h4></h4>`, 227–228
`<h5></h5>`, 227–228
`<h6></h6>`, 227–228
`<head></head>`, 228–229
HTML, 2–4
`<html></html>`, 229
`<img />`, 230
`<input />`, 231–232
`<ins></ins>`, 232–233
`<kbd></kbd>`, 233
`<label></label>`, 233–234
`<legend></legend>`, 234
`<li></li>`, 234–235
`<link>`, 235–236
`<map></map>`, 236–237
`<meta />`, 237–238
`<noscript></noscript>`, 238
`<object></object>`, 150–151, 239–240
`<ol></ol>`, 240
`<optgroup></optgroup>`, 241
`<option></option>`, 241–242

<param />, 243–244
<pre></pre>, 244
<q></q>, 244–245
<samp></samp>, 44–46, 245
<script></script>, 246
<select></select>, 247
, 16–17, 248
, 248–249
<style></style>, 249–250
<table></table>, 250–251
<tbody></tbody>, 251–252
<td></td>, 252–254
<textarea></textarea>, 254–255
<tfoot></tfoot>, 255–256
<th></th>, 256–258
<thead></thead>, 258–259
<title></title>, 259–260
<tr></tr>, 260–261
, 136–138, 261–262
<tbody></tbody> tag, 251–252
<td></td> tag, 252–254
text
images replacement, 88–92
structuring, 37–39
abbreviations, 43–44
acronyms, 43–44
addresses, 47
bidirectional text, 47
editorial insertions and deletions, 46–47
emphasis, 39–40
headings, 40–41
line breaks, 39–40
multilanguage text, 47
paragraphs, 39–40
preformatted text, 44–46
quotations, 42–43
styling
bold, 54–55
color, 50
fonts, 48–49
horizontal alignment, 58
indenting, 58
italics, 54–55
line height, 53–54
shorthand properties, 55–56
size, 50
spacing, 57
strikethroughs, 56–57
techniques, 60
underlines, 56–57
upper and lower case, 55
vertical alignment, 59
text-align property, 58, 313–314
text attribute, input element, 176–177
text-decoration property, 56–57, 314
text-indent property, 315
text-transform property, 315–316
textarea element, form fields, 182–183
<textarea></textarea> tag, 254–255
<tfoot></tfoot> tag, 255–256
<th></th> tag, 256–258
<thead></thead> tag, 258–259
title attribute, 4–7, 70–71
structuring text, 42
tag application, 207–208
title element, 8
<title></title> tag, 14–15, 259–260
titles, links, 70–71
top property, 316
<tr></tr> tag, 156–158, 260–261
transitions, XHTML, 10

U

 tag, 136–138, 261–262
underlines, styling text, 56–57
unicode-bidi property, 316–317
units, CSS values, 25–27
Universal Resource Locator (URLs), links, 62
universal selectors, 24
unordered lists, 136–138
upper case text, styling, 55
URLs (Universal Resource Locator), links, 62
usemap attribute, 82
user style sheets, 28–31

V

values
absolute, 51–52
CSS, 25–31
padding, 98
relative, 51–52
vertical-align property, 59, 317–318
vertical alignment, styling text, 59

visibility property, 318–319
:visited pseudo-class, 268–269
visited states, 66
Vivabit website, 76

W

W3C website, xviii
Web standards, xviii–xix, xxiii–xxix
Webb, Dan, 116
webpages
    CSS. *See also* CSS
        applying to HTML, 32–34
        properties, 23–25
        rules, 17
        selectors, 18–23
        values, 25–31
    HTML, 1. *See also* HTML
        attributes, 2–4
        basic document structure, 8–16
        core attributes, 4–7
        elements, 2–4
        event attributes, 8
        il8n attribute, 7
        tags, 2–4
websites
    Cork'd, 172
    CSS Zen Garden, 89, 118
    Digital Web Magazine, 135
    HTML Dog, 5
    @media 2006, 38
    Vivabit, 76
    W3C, xviii
white-space property, 319–320
widows property, 320
width attribute, img element, 80
width property, 95–97, 320–321
word-spacing property, 57, 321

X

XHTML (eXtensible HTML), xvii. *See also* HTML
    comments, 13
    tags, 2
XHTML Transitional, 10
xml:long attribute, HTML document language declaration, 11

Z

z-index property, 110, 322